Semantic Caching for OLAP via LLM-Based Query Canonicalization

Laurent Bindschaedler[1,*]

[1]Max Planck Institute for Software Systems, Saarbrücken, Germany

Abstract

Analytical workloads exhibit substantial semantic repetition, yet most production caches key entries by SQL surface form (text or AST), fragmenting reuse across BI tools, notebooks, and NL interfaces. We introduce a safety-first middleware cache for dashboard-style OLAP over star schemas that canonicalizes both SQL and NL into a unified key space—the *OLAP Intent Signature*—capturing measures, grouping levels, filters, and time windows. Reuse requires exact intent matches under strict schema validation and confidence-gated NL acceptance; two correctness-preserving derivations (roll-up, filter-down) extend coverage without approximate matching. Across TPC-DS, SSB, and NYC TLC (1,395 queries), we achieve 82% hit rate versus 28% (text) and 56% (AST) with zero false hits; derivations double hit rate on hierarchical queries.

Keywords

OLAP, Caching, Canonicalization, Signatures, Semantics, Star-schema, Natural language, LLMs

1. Introduction

Interactive analytics systems increasingly serve diverse clients: BI dashboards, notebooks, and text-to-SQL interfaces accepting natural language (NL) queries. These use cases amplify latency and compute concerns as LLMs are embedded deeper into analytical pipelines [1]. Query caching is a natural lever because OLAP queries scan large fact tables but return small aggregated results [2], making results inexpensive to materialize and reuse.

Analytical caches typically key entries by SQL surface form: normalized query text or Abstract Syntax Tree (AST)-derived canonical representations [3]. NL clients must first translate to SQL, then apply conventional caching. These approaches fragment reuse across heterogeneous clients (Section 2): the same OLAP question issued from different tools or phrased differently in NL generates distinct cache keys, causing redundant backend execution.

This paper presents a semantic caching middleware designed for a specific category of queries: dashboard-style OLAP aggregations over star or snowflake schemas with a single fact table and dimension joins. We focus on this well-defined OLAP subset (Section 3.1) to make equivalence checkable; queries outside this subset bypass the cache and execute directly on the backend. The main idea is to canonicalize both SQL and NL into a shared, structured cache key, known as an *OLAP Intent Signature*. This signature captures the semantics that affect the numeric result: measures, grouping levels, filters, time windows, and post-aggregation operators. SQL is mapped deterministically from the AST, while NL is translated using a large language model (LLM) constrained to a JSON schema.

The design prioritizes correctness. By default, cache reuse aims for *exact-intent* hits under strict validation. To improve hit rates without relying on approximate matching, we introduce two safe derivations with explicit preconditions: (i) rolling up from finer-grain cached results for additive measures [4], and (ii) filtering down from cached supersets when the cached result includes the necessary filter attributes [5]. Intent signatures increase reuse by consolidating SQL variants and NL paraphrases into a single key; derivations extend coverage across drill patterns (roll-up, filter tightening).

Across three workloads (TPC-DS, SSB, NYC TLC), our cache achieves 82% hit rate versus 28.2% (text) and 55.6% (AST), reducing backend compute by 85–90%. Safe derivations raise hit rate from 37% to 80%

DOLAP 2026: 28th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2026, March 24, 2026, Tampere, Finland

[*] Corresponding author.

bindsch@mpi-sws.org (L. Bindschaedler)
https://binds.ch (L. Bindschaedler)
0000-0003-0559-631X (L. Bindschaedler)

on hierarchical workloads. Although NL canonicalization accuracy degrades under ambiguity (44% adversarial, 51% BIRD [6]), layered safety mechanisms prevent incorrect reuse. At a 0.5 confidence threshold, precision reaches 76.9% at 36.5% coverage.

This paper makes the following contributions:

- An *OLAP Intent Signature* that canonicalizes heterogeneous SQL and NL into a unified cache key space, enabling cross-client reuse without requiring a single semantic API.
- Safety-first reuse with schema validation and confidence gating, with quantified NL failure modes (44% adversarial accuracy) showing why safety mechanisms are essential.
- Safe derivations (roll-up, filter-down) that extend reuse without approximate matching.

2. Background and Motivation

Most production query caches key entries by SQL surface form (normalized text or AST-derived representations) [3]. This fragments reuse: heterogeneous clients (BI tools, notebooks, NL interfaces) generate different SQL for identical questions. The result is *high semantic repetition without cache reuse*. We focus on *query-level* caching (complete results), not cell-level OLAP caches that store individual cube cells [7]; our middleware approach works with any SQL-compatible backend. Our key observation is that dashboard-style OLAP queries over star schemas share stable semantic structure [2]: measures, grouping levels, filters, and time windows. We key the cache by an *OLAP Intent Signature* encoding these components, enabling cross-client and cross-modality reuse.

3. System Design

This section describes a middleware cache that answers OLAP-style aggregation queries. It maps each request (SQL or NL) to a structured *OLAP Intent Signature*, the cache key. The design is correctness-first: reuse requires exact intent equality with strict validation. Extensions beyond exact matches are limited to correctness-preserving transformations under stated assumptions.

3.1. Scope and Assumptions

We focus on a high-value class of dashboard queries, specifically aggregations over a star or snowflake schema [2] with a single fact table and joins to dimension tables along schema-defined foreign keys. The queries may consist of `WHERE`, `GROUP BY`, `HAVING`, and optionally `ORDER BY` and `LIMIT`. We deliberately exclude features that complicate semantic equivalence, such as window functions, set operations, correlated subqueries, recursive CTEs, and lateral joins.

Scope Coverage. In TPC-DS, only 14% of queries qualify (the rest use window functions, CTEs, or set operations); SSB and NYC TLC are 100% covered. Queries outside the scope of supported operations bypass the cache and execute directly on the backend.

To make equivalence checkable, we assume that join semantics are dictated by the schema: given the fact table and referenced dimension attributes, there is a unique join path.

Terminology. We adopt standard OLAP terminology [8]: a *dimension* is a conceptual grouping (e.g., Time, Geography), while a *level* is a specific granularity within a dimension hierarchy (e.g., Year > Quarter > Month). *Roll-up* aggregates from finer to coarser levels; *drill-down* moves from coarser to finer levels. *Slicing* fixes a single dimension value; *dicing* selects ranges across multiple dimensions.

3.2. Architecture

Figure 1 illustrates the system architecture. The middleware operates as an intermediary between clients, such as BI tools, notebooks, and NL interfaces, and the backend OLAP engine.

For each request, the following steps are taken: (1) canonicalize the request into an intent signature, (2) validate the signature against the schema and safety rules, (3) look up the signature hash in the cache, (4) on a miss, execute on the backend and store the result under the signature. Because canonicalization

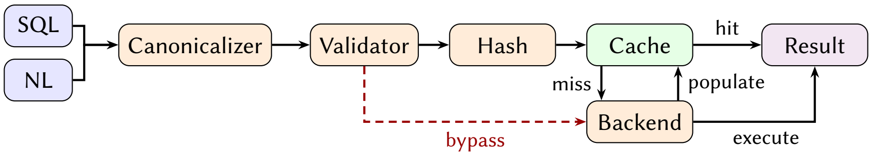

Figure 1: Architecture: queries are canonicalized, validated, and hashed as cache keys. Hits return cached results; misses execute on backend. Validation failures bypass the cache (dashed path). Derivation checks (roll-up, filter-down) occur within cache lookup (not shown).

and validation are distinct steps, reuse decisions are auditable: the signature is an explicit contract that determines whether a cached result may be returned.

3.3. OLAP Intent Signature

We define a canonical signature as a JSON object containing all semantics that can affect the numerical output of the query. The components of the signature include:

- **Measures**: aggregation function and base expression (for example `SUM(f_sales)`), including `DISTINCT` flags.
- **Grouping levels**: list of hierarchy levels in `GROUP BY`, sorted alphabetically to a canonical order (for example `geo.region`, `time.month`). SQL GROUP BY order is semantically irrelevant; we canonicalize to ensure equivalent queries produce identical signatures.
- **Filters**: a set of normalized predicates over non-temporal dimensions and facts, including canonical literal formats.
- **Time window**: explicit start and end boundaries on the time dimension, normalized to a canonical representation. We separate time windows from general filters for two reasons: (i) temporal predicates require special canonicalization (resolving "last quarter" to concrete dates, handling timezone normalization), and (ii) time boundaries are critical for cache invalidation—when new data arrives, entries with open-ended time windows ("last 30 days") must be refreshed while closed windows ("Q1 2024") remain valid.
- **Post-aggregation operators**: `HAVING`, `ORDER BY`, `LIMIT`.
- **Metric identity (optional)**: a metric identifier from a governed layer, if available.
- **Scope (optional)**: for multi-tenant deployments, a tenant or user identifier ensures cache isolation across security boundaries.

The optional fields are relevant for governed or multi-tenant deployments; our evaluation uses ungoverned single-tenant schemas.

We serialize the signature into a canonical JSON string (with sorted keys and normalized lists) and compute a hash (SHA-256) to obtain a fixed-length cache key. This ensures different surface forms map to the same signature (Figure 2). Join paths are implicit, determined by the schema's foreign keys, so signatures omit them; queries with ambiguous joins (role-playing dimensions, self-joins) bypass caching.

NL: "Show total revenue by region for electronics in Q1 2024"

SQL:

```
SELECT r.region_name,
     SUM(s.amount) AS revenue
FROM sales s
  JOIN regions r ON s.region_id = r.id
  JOIN products p ON s.product_id = p.id
WHERE p.category = 'electronics'
  AND s.sale_date >= '2024-01-01'
  AND s.sale_date < '2024-04-01'
GROUP BY r.region_name
```

OLAP Intent Signature:

```
{
  "measures": [{"agg": "SUM",
    "expr": "sales.amount"}],
  "levels": ["regions.region_name"],
  "filters": [{"col": "products.category",
    "op": "=", "val": "electronics"}],
  "time_window": {
    "start": "2024-01-01",
    "end": "2024-04-01"}
}
```

Figure 2: Example: an NL question and equivalent SQL query produce identical OLAP Intent Signatures.

3.4. Canonicalization

SQL requests are parsed into an AST and normalized deterministically (identifier resolution, predicate ordering, literal canonicalization). NL requests are mapped to signatures using an LLM constrained to produce strict JSON; the LLM also returns a confidence score (0–1) used for safety gating. Low-confidence signatures bypass caching to avoid incorrect reuse.

3.5. Validation and Cache Lookup

Before reuse, we validate that all referenced measures/dimensions exist, time windows resolve to concrete boundaries, and join paths are unambiguous. Validation failures bypass the cache and execute on the backend, prioritizing misses over incorrect reuse.

3.6. Correctness-Preserving Reuse Beyond Exact Hits

Exact intent matching is the default reuse mode. We also support two safe derivations:

Roll-Up from Finer-Grained Cache. If a cached entry has identical measures and filters but finer grouping levels, we can re-aggregate the cached result. Roll-up is permitted only for composable aggregations (SUM, COUNT, MIN, MAX); it is rejected for AVG, COUNT DISTINCT, or ratios.

Filter-Down from Cached Supersets. If a cached entry uses a superset filter and contains the attributes needed to apply a tighter filter, we filter the cached result. Derivations are disabled when ORDER BY or LIMIT is present.

Both derivations have explicit preconditions that prevent incorrect reuse. Note that drill-down (finer ← coarser) is not supported: query-level caching lacks the detailed data needed to derive finer-grained results from coarser aggregates.

3.7. Safety Policy for NL-Driven Reuse

NL canonicalization can produce schema-valid but semantically incorrect signatures. We control NL reuse via layered policies: (1) schema validation for structural correctness, (2) confidence-gated reuse bypassing low-confidence signatures, and (3) heuristic checks rejecting common ambiguity patterns (unresolved relative time, underspecified spatial terms). These layers prioritize misses over false hits.

4. Prototype Implementation

We implement the middleware as a Python prototype with three components. The *canonicalizer* parses SQL via `sqlglot` and transforms it into intent signatures through deterministic AST normalization; NL requests use an LLM endpoint emitting schema-constrained JSON with confidence scores. The *validator* checks that referenced measures/dimensions exist and join paths are unambiguous; failures bypass the cache. The *cache store* uses Parquet files indexed by signature hash with a SQLite metadata index for derivation candidate lookup. NL string-to-signature mappings are memoized to avoid repeat LLM calls.

5. Evaluation

5.1. Setup and Results

We evaluate on three OLAP workloads: TPC-DS [9] (14 in-scope queries), SSB [10] (13 queries), and NYC TLC [11] (18 queries). For each of 45 canonical intents, we generate 21 SQL variants (formatting, alias, predicate order changes) and 10 NL paraphrases, yielding 1,395 queries total. All variants are verified to produce identical results, establishing ground truth for hit-rate and false-hit measurement. We compare against TextCache (normalized SQL text), ASTCache (deterministic AST canonicalization), and NL-to-SQL+AST (NL translated to SQL, then AST-cached). For NL robustness, we also test on 63 adversarial queries and 150 BIRD [6] human-authored questions.

Table 1 reports hit rates and reduction factors. LLMSigCache achieves the highest hit rate on all three workloads (82% average), outperforming TextCache (28.2%) and ASTCache (55.6%).

Table 1: Cache performance by method. Hit rate (%); Reduction factor (queries per cache key, higher = less fragmentation). TextCache/ASTCache reduction factors are SQL-only (68% of workload).[†]

	Hit Rate (%)				Reduction (×)		
Method	**NYC TLC**	**SSB**	**TPC-DS**	**Avg**	**NYC TLC**	**SSB**	**TPC-DS**
TextCache	13.8	38.2	32.7	28.2	2.1	3.7	3.2
ASTCache	63.4	54.1	49.3	55.6	32.7	31.0	24.1
NL-to-SQL+AST	86.6	80.4	65.0	77.3	9.4	16.1	8.0
LLMSigCache	96.9	81.9	67.1	82.0	19.0	16.8	9.0

[†]ASTCache excludes NL queries (32% of workload), inflating its per-key ratio on the SQL subset.

Within LLMSigCache, NL reuse arises from both *NL-to-NL* hits (paraphrases of the same intent) and *cross-surface* hits (NL matching SQL-populated entries or vice versa). Cross-surface sharing accounts for 3–34% of NL cache hits; most NL benefit comes from unifying paraphrases under intent keys.

All methods produce **zero false hits** on controlled workloads. NL accuracy drops to 44% on adversarial queries and 51% on BIRD [6] human-authored questions, making safety essential: a 0.5 confidence threshold yields 77% precision at 37% coverage. Backend compute drops 85–90%. Safe derivations (roll-up, filter-down) raise hit rate from 37% to 80% on hierarchical workloads.

5.2. RQ2: Does Intent-Signature Caching Preserve Correctness?

NL Semantic Accuracy under Ambiguity. NL canonicalization accuracy degrades under realistic ambiguity. On 63 adversarial queries (Table 2), accuracy is 44.4%; on 150 human-authored BIRD [6] questions, 51.3%. Time references ("last month") and dimension terms ("area") are most error-prone.

Table 2: Semantic accuracy on 63 adversarial NL queries by ambiguity type.

Ambiguity Type	Example	N	Correct	Wrong	Invalid
Metric name	"revenue" → `net_amount` vs `gross_amount`	15	11	4	0
Time reference	"last month" without current date context	12	2	10	0
Dimension	"area" → `zone` vs `borough`	12	3	9	0
Aggregation	"average trips" → AVG vs COUNT	9	6	3	0
Compositional	Multiple measures in single query	15	6	4	5
Total	—	63	28	30	5

Table 3 shows safety mechanism effectiveness: a 0.5 confidence threshold yields 76.9% precision at 36.5% coverage; schema-specific heuristics improve precision from 48.3% to 69.0%. Model choice matters: Claude-3.5-haiku achieves 60.3% accuracy on adversarial queries versus GPT-4o-mini's 44.4%.

Table 3: Safety mechanism effectiveness on adversarial queries (N=63).

(a) Confidence threshold

Threshold	Coverage	Precision
0.3	63.5%	62.5%
0.5	36.5%	76.9%
0.7	20.6%	78.3%
0.9	12.7%	100.0%

(b) Schema-specific heuristics

	Validation only	With heuristics
Precision	48.3%	69.0%
Wrong signatures	30	9
Bypass rate	7.9%	54.0%

5.3. RQ3: What Are the Backend Savings and Overheads?

SQL cache lookup adds 9–16 ms median overhead (Table 4a). NL canonicalization costs ~1.3 s on first occurrence but is memoized for repeats. Dashboard-like query orderings (Sequential, Zipf) maintain high hit rates even at 10–25% cache capacity (Table 4b).

5.4. RQ4: Do Correctness-Preserving Derivations Extend Coverage?

We construct an SSB hierarchical workload where queries drill through time, geography, and product hierarchies. TPC-DS and NYC TLC lack systematic hierarchy traversal, so derivations provide limited

Table 4: RQ3 overhead measurements.

(a) Latency (ms) by scenario

Scenario	Med.	P95
SQL queries (all methods)		
Cache lookup	9–16	12–28
NL queries (LLMSigCache)		
First occur. (miss)	1,317	2,266
First occur. (hit)	1,304	2,016
Repeat (memo)	<0.01	<0.01

(b) Hit rate (%) vs. cache size (NYC TLC)

Ordering	10%	25%	50%	75%	100%
Sequential	96.8	96.8	96.8	96.8	96.9
Random	4.9	22.9	47.5	69.9	96.9
Interleaved	0.0	5.1	5.1	5.1	96.9
Zipf	13.4	45.2	72.8	85.7	96.9

benefit on those workloads. Without derivations, hit rate is 37%; enabling roll-up and filter-down raises it to 80% with zero false hits.

6. Discussion

Cache Invalidation. Our evaluation uses static snapshots, but production systems must link entries to data freshness [12]. Queries with closed time windows (“Q1 2024”) remain valid indefinitely; open-ended windows (“last 30 days”) require refresh. Empirical characterization of cache churn under realistic update patterns is left for future work.

Deployment Configurations. The precision-coverage trade-off is tunable: conservative (threshold 0.7, all heuristics) yields 71.4% precision at 22.2% coverage; balanced (0.5, time+spatial) reaches 72.0% at 39.7%; disabling safety drops to 48.3% precision with 30 wrong results.

Limitations. Key limitations include: (1) queries with window functions, set operations, or CTEs bypass the cache; in TPC-DS, only 14% of queries qualify, though dashboard-oriented workloads (SSB, NYC TLC) are fully covered; (2) NL canonicalization accuracy drops on ambiguous inputs (44% adversarial, 51% BIRD [6]), requiring confidence gating and heuristic checks; (3) roll-up derivations are limited to additive measures; and (4) synthetic workloads may overstate hit rates versus production traffic. Cache invalidation under data updates remains the primary systems gap for production deployment.

7. Related Work

Classic semantic caching [13, 14] stores results by predicate regions. We use a structured *intent signature* derivable from both SQL and NL. Prior work on aggregate views [5, 4] and summarizability [15, 16] informs our roll-up derivations. Text-to-SQL research [17, 6, 18, 19] enables NL→SQL→cache pipelines, but translation variability fragments reuse; we canonicalize NL directly to intent signatures. BI semantic layers [20, 21] cache within platform-specific models; we provide portable middleware across heterogeneous clients. Embedding-based caches [22] use similarity but cannot ensure correctness.

8. Conclusion

We introduced a safety-first semantic cache for dashboard-style OLAP that canonicalizes SQL and NL into a unified key space, the *OLAP Intent Signature.* Evaluation shows large hit-rate and backend-savings gains over text and AST caching, while safety mechanisms prevent incorrect reuse despite NL errors. Key directions for future work include robust cache invalidation under data updates, stronger schema grounding for NL canonicalization, and extending input modalities to MDX.

Declaration on Generative AI

The LLMs evaluated in this work (GPT-4o-mini, Claude-3.5-haiku) are components of the proposed system and were not used to prepare the manuscript. The author used Claude (Anthropic) and Grammarly for writing style, grammar, spelling, and formatting, and OpenAI Deep Research for citation management. The author reviewed and edited all outputs and takes full responsibility for the content.

References

[1] B. Mohammadi, L. Bindschaedler, The case for instance-optimized LLMs in OLAP databases, in: Proceedings of DOLAP 2025: 27th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2025, volume 3931 of *CEUR Workshop Proceedings*, CEUR-WS.org, Barcelona, Spain, 2025, pp. 59–63. URL: https://ceur-ws.org/Vol-3931/short3.pdf.

[2] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals, Data Mining and Knowledge Discovery 1 (1997) 29–53. URL: https://doi.org/10.1023/A:1009726021843. doi:10.1023/A:1009726021843.

[3] ClickHouse, Query cache, ClickHouse Docs, 2026. URL: https://clickhouse.com/docs/operations/query-cache, last modified 2026-01-15 (Git history). Accessed 2026-02-20.

[4] C.-S. Park, M.-H. Kim, Y.-J. Lee, Rewriting OLAP queries using materialized views and dimension hierarchies in data warehouses, in: Proceedings of the 17th International Conference on Data Engineering (ICDE 2001), Heidelberg, Germany, April 2–6, 2001, IEEE Computer Society, 2001, pp. 515–523. URL: https://doi.org/10.1109/ICDE.2001.914865. doi:10.1109/ICDE.2001.914865.

[5] D. Srivastava, S. Dar, H. V. Jagadish, A. Y. Levy, Answering queries with aggregation using views, in: Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB '96), Mumbai (Bombay), India, September 3–6, 1996, Morgan Kaufmann, 1996, pp. 318–329. URL: https://www.vldb.org/conf/1996/P318.PDF.

[6] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. Chang, F. Huang, R. Cheng, Y. 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. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

[7] J. Hyde, Cache control, Pentaho Mondrian Documentation, 2011. URL: https://mondrian.pentaho.com/documentation/cache_control.php, author: Julian Hyde; last modified by Luc Boudreau, August 2011. Accessed 2026-02-20.

[8] M. Golfarelli, S. Rizzi, Data Warehouse Design: Modern Principles and Methodologies, McGraw Hill Professional, 2009.

[9] Transaction Processing Performance Council, TPC Benchmark™ DS Standard Specification, Transaction Processing Performance Council, 2021. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.2.0.pdf, version 3.2.0.

[10] P. O'Neil, B. O'Neil, X. Chen, Star Schema Benchmark, Technical Report, University of Massachusetts Boston, 2009. URL: https://www.cs.umb.edu/~poneil/StarSchemaB.PDF, revision 3.

[11] New York City Taxi and Limousine Commission, TLC trip record data, NYC Taxi & Limousine Commission website (NYC.gov), 2026. URL: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page, accessed 2026-02-20.

[12] A. Gupta, I. S. Mumick, Maintenance of materialized views: Problems, techniques, and applications, IEEE Data Engineering Bulletin 18 (1995) 3–18. URL: https://dblp.org/rec/journals/debu/GuptaM95.html.

[13] S. Dar, M. J. Franklin, B. Þ. Jónsson, D. Srivastava, M. Tan, Semantic data caching and replacement, in: Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB '96), Mumbai (Bombay), India, September 3–6, 1996, Morgan Kaufmann, 1996, pp. 330–341. URL: https://www.vldb.org/conf/1996/P330.PDF.

[14] P. Scheuermann, J. Shim, R. Vingralek, WATCHMAN: A data warehouse intelligent cache manager, in: Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB '96), Mumbai (Bombay), India, September 3–6, 1996, Morgan Kaufmann, 1996, pp. 51–62. URL: https://www.vldb.org/conf/1996/P051.PDF.

[15] H.-J. Lenz, A. Shoshani, Summarizability in OLAP and statistical data bases, in: Proceedings of the 9th International Conference on Scientific and Statistical Database Management (SSDBM '97),

Olympia, Washington, USA, August 11–13, 1997, IEEE Computer Society, 1997, pp. 132–143. URL: https://doi.org/10.1109/SSDM.1997.621175. doi:10.1109/SSDM.1997.621175.

[16] T. B. Pedersen, C. S. Jensen, C. E. Dyreson, Extending practical pre-aggregation in on-line analytical processing, in: Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99), Edinburgh, Scotland, UK, September 7–10, 1999, 1999, pp. 663–674. URL: https://www.vldb.org/conf/1999/P62.pdf.

[17] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, D. Radev, Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018), Brussels, Belgium, October 31–November 4, 2018, 2018, pp. 3911–3921. URL: https://aclanthology.org/D18-1425/. doi:10.18653/v1/D18-1425.

[18] M. Pourreza, D. Rafiei, DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction, in: Advances in Neural Information Processing Systems, volume 36, 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/72223cc66f63ca1aa59edaec1b3670e6-Abstract-Conference.html.

[19] S. Chaturvedi, A. Chadha, L. Bindschaedler, SQL-of-thought: Multi-agentic text-to-SQL with guided error correction, in: Deep Learning for Code in the Agentic Era (NeurIPS 2025 Workshop: DL4C), 2025. URL: https://arxiv.org/abs/2509.00581.

[20] Cube, Introduction, Cube Documentation, 2026. URL: https://cube.dev/docs/product/introduction, last modified 2026-02-19 (Git history). Accessed 2026-02-20.

[21] dbt Labs, Inc., dbt semantic layer, dbt Developer Hub, 2026. URL: https://docs.getdbt.com/docs/use-dbt-semantic-layer/dbt-sl, last updated 2026-02-19. Accessed 2026-02-20.

[22] F. Bang, GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings, in: L. Tan, D. Milajevs, G. Chauhan, J. Gwinnup, E. Rippeth (Eds.), Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023), Association for Computational Linguistics, Singapore, 2023, pp. 212–218. URL: https://aclanthology.org/2023.nlposs-1.24/. doi:10.18653/v1/2023.nlposs-1.24.