International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(15s) 2026 pp. 132-149
https://doi.org/10.70917/ijcisim-2026-4353
Article

Evaluation of Large Language Models for Natural Language to SQL Query Generation: A Comparative Study Using Exact Match and Execution Accuracy

Bharath Kumar N[1*], T L Manasa[2]

[1*]IBM, ORCID: https://orcid.org/0009-0000-4808-1784

[2]Faculty of Engineering & Technology, Department of Electrical & Electronics Engineering, Mangalayatan University, Beswan, Aligarh, UP, India. ORCID: https://orcid.org/0009-0005-0414-7119

Abstract: Decisions are being made based on data, and the demands for easy-to-use interfaces that enable users to simply type in a question to a relational database without having to be a Structured Query Language (SQL) expert have increased. Although the development of Large Language Models (LLMs) has seen remarkable advancements in the past few years, and has led to the emergence of LLMs that have significantly improved the generation of SQL from natural language (Text-to-SQL)[1], there is a relative lack of systematic comparisons between the latest commercial LLMs that has been tested against a series of graded SQL complexity levels with both syntactic and execution-based metrics. This research aims to assess and contrast ChatGPT, Google Gemini, DeepSeek, and Microsoft Copilot for their accuracy and efficacy in creating SQL queries from natural language queries. In a quantitative experimental design, 100 natural language questions were given using a purpose-built benchmark, with six types of queries that included simple retrieval, filtering, joins, aggregation, GROUP BY and HAVING, and nested subqueries. The results of the models were assessed using the manually written and validated ground truth query, with the use of normalised Exact Match Accuracy (EMA) and Execution Accuracy (EXA). The best overall results (EMA 97%, EXA 97%) were obtained by ChatGPT. The Copilot scored EMA 56% and EXA 96%, DeepSeek EMA 55% and EXA 93% and Gemini the lowest with a score of EMA 36% and EXA 87%. The large margin between EMA and EXA for the models (+51 points, +40 points, +38 points) demonstrates the significant shortcomings of exact-match scoring when it comes to assessing real-world skills for producing valid but structurally different SQL. The study empirically demonstrates comparative behaviours of the current LLMs, and suggests the use of execution-aware evaluation in the studies and deployment of Text-to-SQL.

Keywords: Text-to-SQL; Large Language Models; ChatGPT; Gemini; DeepSeek; Microsoft Copilot; Execution Accuracy; Exact Match Accuracy; query complexity; evaluation methodology.

1. Introduction

Data-driven decision making has completely revolutionized the way organizations work, manage and innovate[2]. Many databases are relational and house important information, such as in retail, healthcare, finance, and manufacturing, that are used to run business processes. Such systems are commonplace, but a challenge to access the information they have is the continued failure to write accurate SQL queries[3]. SQL is a common and powerful language for the database professional, but what if a business analyst, manager, domain expert or non-technical user has an information requirement that can only be met with a query that requires expertise to formulate? The topic of human language to database language has been inspiring the study of Natural Language Interfaces to Database (NID) for years, especially of Natural Language to SQL (Text-to-SQL) (T2S) systems.

With the rise of LLMs, the definition and scope of Text-to-SQL systems have shifted.Thanks to LLMs, the definition and scope of Text-to-SQL has evolved. Whereas the previous generation of machine learning models struggled to interpret complex instructions, generate executable code and do reasoning, models like ChatGPT, Gemini, DeepSeek and Microsoft Copilot are able to do all three things. Modern LLM models can be used to generate SQL directly from natural language prompts, without requiring rule-based or template-based approaches.Unlike the rules/templated ones, the models in use today can now abstract the information across different domains and produce SQL directly from a natural language prompt. As a result, LLM-powered interfaces are becoming a more common feature in organisations to make data more accessible, less reliant on specialists and quicker to analyze.

Ideally, a user may enter a query, such as "Which customers made the most money this quarter?" and receive a correct answer without needing to understand the schema, to join tables, use aggregation functions, or construct a nested query. In fact, performance is unfair[2]. A simple filter operation can be executed by a model, and then be inefficient on a complex joins-multiple-tables operation, a sum or a nested subquery. More subtly, two models can yield completely different SQL statements which end up with the same results, leading to the methodological question that this paper addresses: How can Text-to-SQL systems be evaluated and compared?

The development of systems that are evaluated in the light of benchmark data sets such as WikiSQL, Spider, BIRD has seen significant progress in a relatively short period of time [5, 6]. The early neural models were sequence-to-sequence models and schema linking, and the recent ones were utilizing transformer-based models and prompt engineering[3, 7]. Overall, the findings from survey studies show that “LLM-generated SQL has improved the accuracy of SQL over previous methods[1]. However, there are some things that haven't been clarified. Many studies have been conducted using benchmarks which do not reflect real enterprise databases[8] [9]. Secondly, most studies investigate individual models or individual prompting techniques rather than providing comparisons across multiple recent LLMs using a set of parameters [10]. Specifically in this work, it is important to consider the concept of Exact Match Accuracy, which implies that multiple semantically similar queries may produce the same answer, but an answer generated by the system matches a reference query[11][12].

In practice these restrictions have consequences. When using generative AI tools for analytics, decisions need to be made around the choice of model, trustworthiness, and governance. Inaccuracy in SQL can result in wrong business information and loss of trust, over-optimism of the use of exact-match scoring can create an underestimation of models that can be valid alternate formulations[11]. It's a technical and operational problem, then, to know if a modern LLM is effective on a given level of SQL difficulty and following a given evaluation metric.

In this paper, we compare four widely used LLMs - ChatGPT, Google Gemini, DeepSeek, and Microsoft Copilot with 100 natural language questions across six categories of SQL tables based on two metrics: Exact Match Accuracy and Execution Accuracy. The work is based on the notion that good Text-to-SQL systems should be both syntactically correct and semantically and execution-level correct, and uses a dual perspective to differentiate between textual similarity and result correctness.

1.1 Objectives of the Study

The study pursues the following objectives:

1. To assess the ability of ChatGPT, Gemini, DeepSeek, and Microsoft Copilot to translate natural language questions into executable SQL queries.
2. To compare the performance of these models using Exact Match Accuracy and Execution Accuracy.
3. To analyse the impact of SQL query complexity on model performance across simple, filtering, join, aggregation, group-by/having, and nested-subquery categories.
4. To identify the strengths and weaknesses of each model in handling different SQL query types, supported by a structured analysis of failure modes.
5. To examine whether Execution Accuracy provides a more meaningful evaluation of Text-to-SQL systems than Exact Match Accuracy, and to characterise the disagreement between the two metrics.

1.2 Significance and Organisation

The paper offers empirical evidence of the comparative performance of recent LLMs in Text-to-SQL tasks and aids in further discussion about the proper means of assessing them. The implications of the results in practice are clear in model selection and model deployment strategies of natural-language database interface. The rest of the paper summarizes related work (Section 2), outlines the experimental approach (Section 3), summarizes results and provides figures (Section 4), discusses implications and limitations (Section 5), and concludes (Section 6).

2. Literature Review

Text-to-SQL generation is an intersection of NLP, database, and AI. As the volume of data stored in an organisation continues to increase, it is more important than ever that users can interact with interfaces and query the data in the databases without being familiar with SQL[13]. The traditional querying paradigm assumes the knowledge of schema structure, as well as join, aggregation, and syntax, thus data accessibility is lacking in terms of data availability[14,15]. The arrival of LLM has catapulted the progress towards this objective by making faster advances in bridging this gap, but it has also raised new concerns about comparative effectiveness, reliability, and evaluation[1] and [10].

In the initial work, the rule-based approach, grammar-based approach and template matching[3] were used to implement T2S. These showed good performance on small subsets of data but failed to scale and fail gracefully to new schemas. Deep learning techniques were used to focus on sequence-to-sequence models that learn the translation from natural language to SQL[3] but those approaches were not effective for good generalisation in the presence of complex schemas and semantic ambiguities. This was a turning point as the introduction of WikiSQL and Spider provided standardized evaluation settings and common evaluation metrics, including Exact Match Accuracy and Execution Accuracy[3].

Again, transformer models and later LLMs took over the space. In contrast to the task-specific architectures, LLMs can directly create SQL via in-context learning and prompt engineering[8]. Recent comprehensive surveys provide an overview of recent improvements in prompt design, schema linking, retrieval augmentation, and execution-based evaluation, and, generally, indicate that LLM models have outperformed previous generations of neural models by virtue of their enhanced language capabilities and reasoning power[2]. Most of these reviews are descriptive, however, and offer very little in the way of comparisons of the various models, so no real information can be gleaned from them as to how well the various models operate under similar conditions.

Other studies point out the importance of timely design and fine tuning strategies, as the performance may vary considerably according to the representation of the database structure in the prompt[8]. This kind of writing assists in the implementation of work, and focuses on the methodological refinement, not in head-to-head comparison of models. Likewise, another thread sees LLM as a new generation database interface with a focus on schema linking, contextual reasoning and query complexity as important factors affecting LLM performance and only a few empirical comparisons on different complexity grades[16].

Empirical benchmark studies show that the GPT models consistently outperform smaller language models, especially when using prompt engineering, and indicate that it is important that it is done correctly, not similar to the text[8]. The Spider benchmark was a much more complex benchmark with multi-domain queries that showed that there were large differences between machine-generated SQL and human-written SQL, and promoted more sophisticated models. Models designed for Spider may not be easily translatable to other enterprise systems with different schema and naming conventions, however, since the application is research based. Subsequent benchmarks (e.g., BIRD) have introduced more complex and realistic examples, but studies of such benchmarks continue to rely upon leader board rankings and exact-match results, which may fail to capture the true impact of semantically similar queries that have differing structural variations.

The theme running throughout is thus the duality between execution accuracy and exact match accuracy. ExactMatch is used to check whether a query matches a reference query and ExecutionAccuracy is used to check whether the output of the query generated matches the reference query. However, for several authors, an execution-based evaluation measure is more closer to practical use: even though there are multiple valid evaluation measures in a database, many evaluations are done based on exact-match measures[1] [18]. This lack of uniformity renders the results of the different studies not easily comparable, and this is what this current study is all about.

This work has two other gaps. First, the number of thorough comparative studies of recently released commercial LLMs is relatively limited, and most of the evidence has been limited to GPT-based models or compares them individually[8]. As organisations explore the possibilities of ChatGPT, Gemini, DeepSeek and Copilot for analytics workflows, they will appreciate the value of comparative evidence in controlled environments. Second, there are few studies to date that provide analysis of accuracy differences by query type, and it is difficult to determine where models are successful and where they are failing, and which SQL constructs remain difficult[1].

The literature shows significant advances in Text-to-SQL generation and the transformative power of LLMs, but with some gaps: most of the available evaluations are based on benchmarks, rather than enterprise contexts; the

available evaluations focus on individual models instead of comparing multiple models; the available complexity analysis is done at the model level, rather than at the category level; and the available evaluation measures are exact-match and execution, but not both. The present study fills these gaps by comparing four contemporary LLM models using a custom-made enterprise style benchmark in terms of six categories of model complexity, and by applying a combination of syntactic and semantic features to assess the correctness of the LLM models in a balanced way, using the metrics of Exact Match Accuracy and Execution Accuracy, the latter being measured with normalized values, and complemented with a structured failure-mode analysis.

3. Methods

3.1 Study Design

We used a quantitative experimental design to assess and compare the performance of current LLMs for T2S query generation. The experimental approach was appropriate since the intent was to investigate the capacity of the models under standard and controlled conditions, which insured that the results would be compared directly among the models without the influence of other conditions. To avoid the impact of the ground truth SQL changes in the test time, the natural language questions and the ground truth SQL queries were constructed in April/May 2026, and the four models were tested in the same version of the natural language questions and data. The study was conducted following the well-known principles for conducting machine learning benchmark studies, namely that it should be reproducible, inputs are consistent and there should be objective evaluation criteria.

This study was conducted under controlled conditions, using a custom developed relational database schema and benchmark dataset. To maximise the comparability of the four models, the four models were evaluated during a single evaluation session on 12th June 2026 so that the impact of model updates on comparability is minimised.

3.2 Ethics Statement

The study was conducted without any human subjects, patient information, biological samples or animals; it was performed on a schema and benchmark of synthetic development which were submitted to the LLM test. Ethical approval was not needed in this instance as it was based on institutional research guidelines. Personal data was not gathered, collected or analysed. All systems tested were public facing (wheretowheretod.com) and in use as per their terms and conditions.

3.3 Database Schema and Benchmark Construction

The evaluation testbed consisted of five-table relational schema which include the common entities of an organisation: Customers, Products, Orders, Order_Items and Employees. The schema has complex queries possible as it has primary keys, foreign-key relationships and a self-referencing manager relationship in its Employees table. The schema is illustrated in the figure 1.

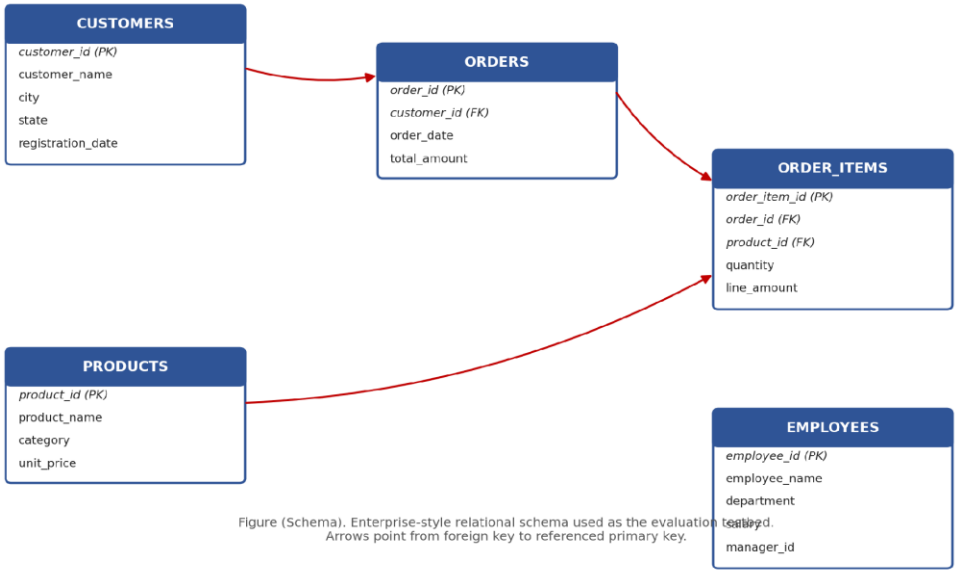

Figure 1. Enterprise-style relational schema used as the evaluation testbed.

Note. Arrows point from each foreign key to the referenced primary key.

A set of 100 natural language questions was designed to simulate real enterprise queries and it was subdivided into six groups of increasing SQL complexity. Unambiguous questions that could be converted to a valid SQL statement were included and questions were not too specific to the domain to be comparable to each other". A ground-truth SQL query was hand written and tested by execution for each question beforehand to guarantee the correctness of the query. Questions are listed by category in Table 1.

Table 1

Distribution of Benchmark Questions by SQL Complexity Category

Category	No. of Questions	Representative SQL Constructs
Simple retrieval	20	SELECT, FROM
Filtering	20	WHERE, comparison/logical operators
Join	20	INNER/OUTER JOIN across tables
Aggregation	20	COUNT, SUM, AVG, MIN, MAX
Group By/Having	10	GROUP BY, HAVING
Nested/Subquery	10	Subqueries, correlated subqueries
Total	100	—

Note. The Group By/Having and Nested/Subquery categories contain 10 questions each; percentages for these two groups therefore move in 10-point increments, a point revisited in the limitations.

3.4 Models Evaluated

Four LLMs were assessed by purposive sampling because they are commonly mentioned in the academic and industrial research and development community in recent years: ChatGPT, Google Gemini, DeepSeek, and Microsoft Copilot. These are not model checkpoints, but products that are changing and all four were accessed with their public web interfaces and queried on 12 June 2026 using the current public version of each (Table 1a) for that date, which is reported to enable reproducibility.

Table 1a

Evaluated Models, Versions, and Access

Model	Version (as accessed)	Access date	Interface
ChatGPT	GPT-5.5 (OpenAI)	12 June 2026	Web
Google Gemini	Gemini 3.5 Flash	12 June 2026	Web
DeepSeek	DeepSeek-V3	12 June 2026	Web
Microsoft Copilot	Current public release (no version string exposed)	12 June 2026	Web

Note. Microsoft Copilot does not expose a discrete version identifier through its interface[19]*; the entry records that the then-current public release was used on the access date. Versions for the other models reflect the label displayed in each product's interface on 12 June 2026.*

3.5 Equipment and Materials

The data was stored in Oracle (Oracle SQL) and queries were run on this platform to check generated queries. Results were recorded and organised in Microsoft Excel and descriptive statistics were computed. The main research materials were the benchmark questions and validated ground-truth queries. To ensure all four models were the same to compare, all 100 natural language questions were standardised, and the same fixed set of questions was asked in the same order to all four models giving all four the same inputs. For all the natural language questions, see Appendix B. The questions were not hidden in a schema and instructions wrapper, but instead were entered as a short natural-language question (e.g., “Show customers whose total purchases exceed 50,000.”) as a typical way that a non-technical user might interact with these tools. All four models were able to generate syntactically valid, runnable Oracle SQL for each question, and there were no queries that were unrunnable for each session, the schema being pre-defined.

3.6 Evaluation Metrics

Two complementary metrics were used. The most important measure was Execution Accuracy (EXA): the generated query returned the same set of results as the ground-truth query when executed on the populated database; structural

differences were not considered wrong and queries that returned equivalent answers through such differences were counted as correct. All 4 models were able to produce executable SQL sentences given a question; however EXA was not just checking if the SQL sentence was executable, it was checking that the result-set was correct, rather than simply that the SQL sentence was correct. The other measure, Exact Match Accuracy (EMA), is the percentage of the queries produced that are at the SQL structure level identical to the ground-truth query.

When running this study with the Exact Match, it was done on a character-level (raw string) basis, so that a generated query would be considered an exact match only when it matched in text to the reference query. There was no white space, case, table name or clause order normalization. There was no operator normalization (such as !=, <> etc.). Its strictness was chosen for transparency and ease of replication but has a significant consequence that is crucial to interpreting the results, which is any query that is logically correct but not a literal replica of the reference: It is considered a “mismatch” if it differs in formatting, aliasing, or in the order of commutative clauses. Strict raw-string matching is then a lower bound on the syntactic agreement and it is expected that it will increase the difference between the scores of the Exact Match and the Execution Accuracy. It is specifically denied to be a restriction (see Section 5) and it is the central thesis of this study that the assessment of practical ability for an execution is a more valid measurement of ability than other types of assessment.

3.7 Procedure

The study was carried out in a series of steps. A large language model (LLM) was used to generate the five-table schema from a single prompt (Appendix A) and then reviewed and finalised by the authors, which was used to

create a functional database for execution and validation by adding sample records to the tables. Second, the 100 question based benchmark was developed in six categories (Appendix B) and a ground-truth SQL query was formulated and tested for each question. Thirdly, each of 100 questions was posed individually on each of the 4 models, in the same order for each model, and, regardless of how generated, its resulting SQL was recorded. Fourth, every produced query was given a score for the Exact Match (where the raw string of the produced query matches the raw string of the reference query, ignoring any difference in the strings) and for Execution Accuracy (by executing both queries, and comparing the result sets). Lastly, the results were combined per model and per category to facilitate overall and category analysis. In all of the models, the same types of questions were shown.

3.8 Failure-Mode Coding Scheme

For each of the failed queries (Objective 4), the query was coded according to the coding scheme in Table 2, rather than just marking it as failed. All the failures reported were semantic failures, that is, queries that were able to be executed without error, but failed to return a result set equal to the ground truth - failures were not due to syntax errors, due to incompatibility with the Oracle dialect, or due to the presence of schema elements that were unknown to the query. The coding scheme thus only addresses the semantic error types that were found in the data. Only one failure category was assigned to each wrong query and if there was more than one failure category the one the researcher thought was most responsible for the wrong query result was recorded. Two independent raters used the scheme and any disagreements were settled by discussion.

Table 2

Coding Scheme for Classifying Execution (Result Set) Failures

Failure Category	Definition
Schema-linking error	Wrong table or column selected; misinterpretation of which entity the question refers to, yielding a different result set.
Join error	Incorrect join condition, wrong join type, or an unintended Cartesian product that alters the returned rows.
Aggregation/grouping error	Incorrect or missing aggregate function, or wrong GROUP BY/HAVING logic or grouping granularity.
Nested-logic error	Incorrect subquery construction or correlation, or flawed nesting of conditions that changes the result.
Semantic misinterpretation	Syntactically valid, executable SQL that correctly runs but answers a different question than the one asked.

The category-level failure counts produced with this scheme are reported in Section 4.6. Because failure classification requires inspecting each incorrect query, the counts are supplied from the authors' records rather than computed automatically.

3.9 Statistical Analysis

Descriptive statistics summarised model performance: Exact Match Accuracy and Execution Accuracy were computed as the number of correct responses divided by the number of evaluated questions, expressed as a percentage, overall and by category. To test whether the within-model difference between Execution and Exact Match correctness was statistically significant rather than incidental, McNemar's test for paired nominal data was applied to each model's per-question outcomes under the two metrics, with the significance level set at $\alpha = 0.05$. Because every exact-match (identical-text) query also returned the correct result set, the discordant pairs consisted entirely of queries that were execution-correct but not exact matches; McNemar's test (with continuity correction) was computed on these discordant counts. Analyses were performed in Python (SciPy).

4. Results

4.1 Overall Performance

Table 3 and Figure 2 present overall performance across the 100 questions. ChatGPT achieved the highest scores on both metrics (EMA 97%, EXA 97%). Microsoft Copilot recorded EMA 56% and EXA 96%; DeepSeek EMA 55% and EXA 93%; and Gemini the lowest, EMA 36% and EXA 87%. For three of the four models, Execution Accuracy substantially exceeded Exact Match Accuracy, indicating that these models frequently generated structurally different but result-correct SQL.

Table 3

Overall Performance of Evaluated Large Language Models (N = 100)

Model	EMA (%)	EXA (%)	EXA − EMA
ChatGPT	97	97	0
Gemini	36	87	+51
DeepSeek	55	93	+38
Microsoft Copilot	56	96	+40

Note. EMA = Exact Match Accuracy; EXA = Execution Accuracy. Values are reconciled to the category-level counts reported in Sections 4.2–4.3.

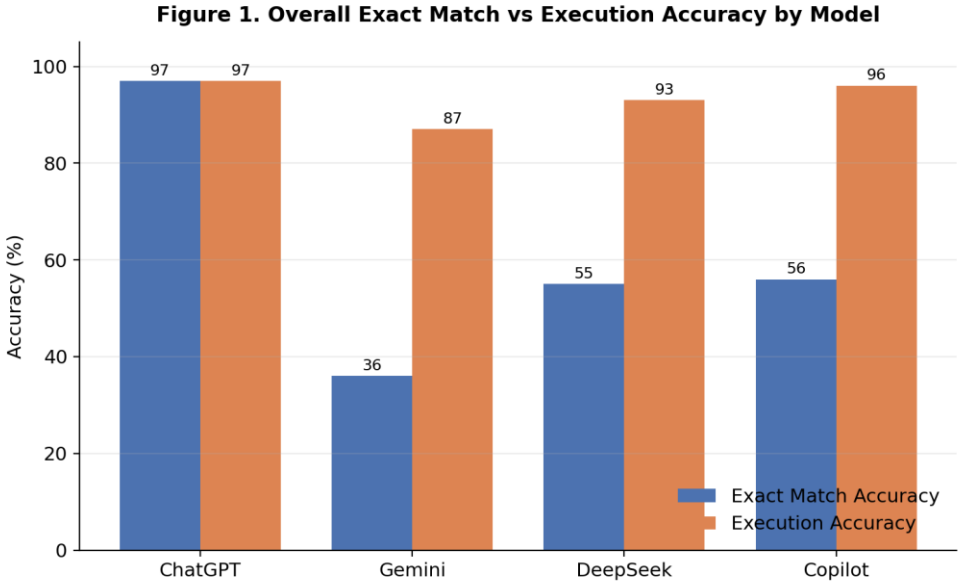

Figure 2. Overall Exact Match Accuracy versus Execution Accuracy by model.

4.2 Category-wise Exact Match Accuracy

Table 4 and Figure 3 report Exact Match Accuracy by category. ChatGPT maintained 100% Exact Match Accuracy in every category except simple retrieval (85%). The other three models showed marked declines as complexity increased. Gemini fell from 45% on simple queries to 10% on group-by/having. Microsoft Copilot declined from 90% on simple queries to 20% on group-by/having and 0% on nested subqueries, despite—as shown next—executing queries in those same categories correctly most of the time.

Table 4

Category-wise Exact Match Accuracy (%)

Category	ChatGPT	Gemini	DeepSeek	Copilot
Simple	85	45	90	90

Filtering	100	55	65	65
Join	100	30	40	60
Aggregation	100	25	40	55
Group By/Having	100	10	40	20
Nested/Subquery	100	40	40	0

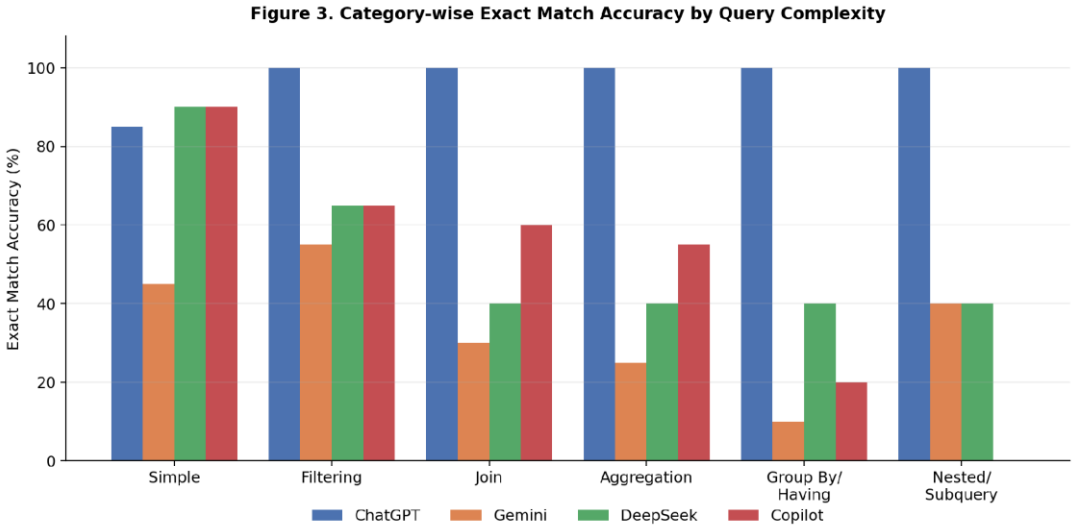

Figure 3. Category-wise Exact Match Accuracy by query complexity.

4.3 Category-wise Execution Accuracy

Table 5 and Figure 4 report Execution Accuracy by category. Filtering queries were handled at or near ceiling by all models. ChatGPT sustained 100% execution accuracy in every category except simple retrieval (85%). Crucially, the three models with low exact-match scores retained high execution accuracy across most categories—for example, Microsoft Copilot executed nested subqueries correctly 90% of the time despite 0% exact match, and GROUP BY and HAVING at 100% despite 20% exact match. Gemini, while lowest overall, still executed most categories at 70% or above, with its weakest results on join and nested-subquery tasks (75% and 70%).

Table 5

Category-wise Execution Accuracy (%)

Category	**ChatGPT**	**Gemini**	**DeepSeek**	**Copilot**
Simple	85	100	100	100
Filtering	100	100	100	100
Join	100	75	75	90
Aggregation	100	90	100	95
Group By/Having	100	70	100	100
Nested/Subquery	100	70	80	90

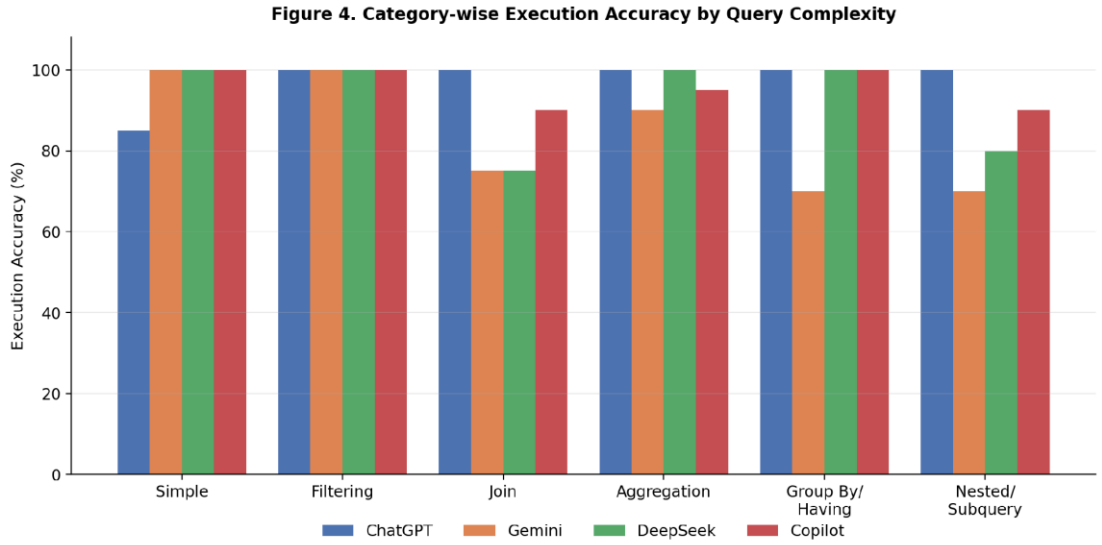

Figure 4. Category-wise Execution Accuracy by query complexity.

4.4 Divergence Between the Two Metrics

The difference between Execution and Exact Match Accuracy is visualised in figure 5. There was no gap for ChatGPT (0 points), which means that the structure of the outputs was very similar to the structure of the reference queries. The widest margins (+51 points) were observed for Gemini, followed by Microsoft Copilot (+40) and DeepSeek (+38). The gaps show that the three of four models had a significant portion of generated queries that, although result-correct, had a different structure from the reference.

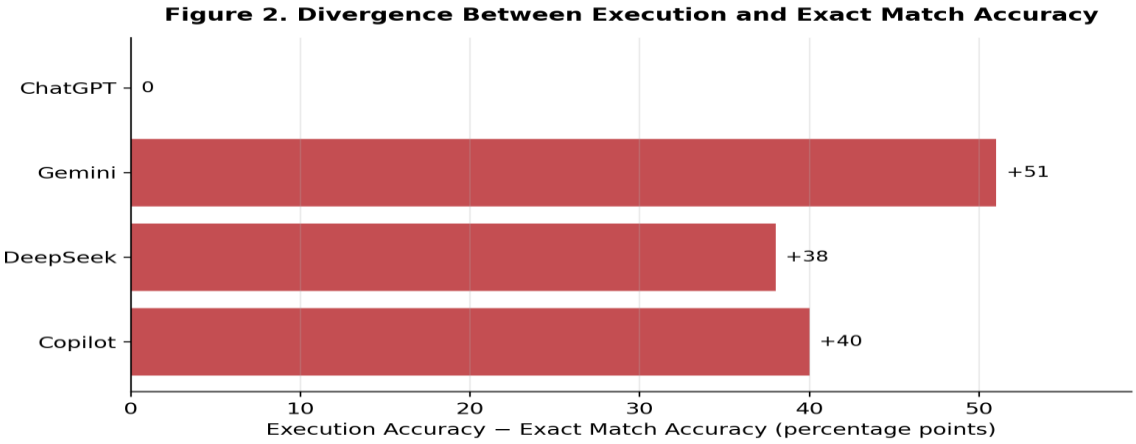

Figure 5. Divergence between Execution and Exact Match Accuracy by model.

There was a significant difference in the within-model divergence of the two measures for the three models with values of McNemar's tests. For Gemini, the difference was significant, $\chi^2(1) = 49.02$, $p < .001$; for Microsoft Copilot, $\chi^2(1) = 38.02$, $p < .001$; and for DeepSeek, $\chi^2(1) = 36.03$, $p < .001$. ChatGPT had no discordant pairs and hence no test was applicable for the two metrics. The results show that the EMA–EXA contrast is unlikely to be due to chance.

4.5 Complexity Degradation

The graph in Figure 6 is Exact Match Accuracy, versus increasing query complexity. While Gemini, DeepSeek and Copilot saw a steady drop in performance with complexity, ChatGPT stayed relatively flat at or around the ceiling for each category. The execution-accuracy curves (Table 5) fell much shallower, which further demonstrated that the models were often able to generate a valid SQL when they were not rewarded using the exact-match metric.

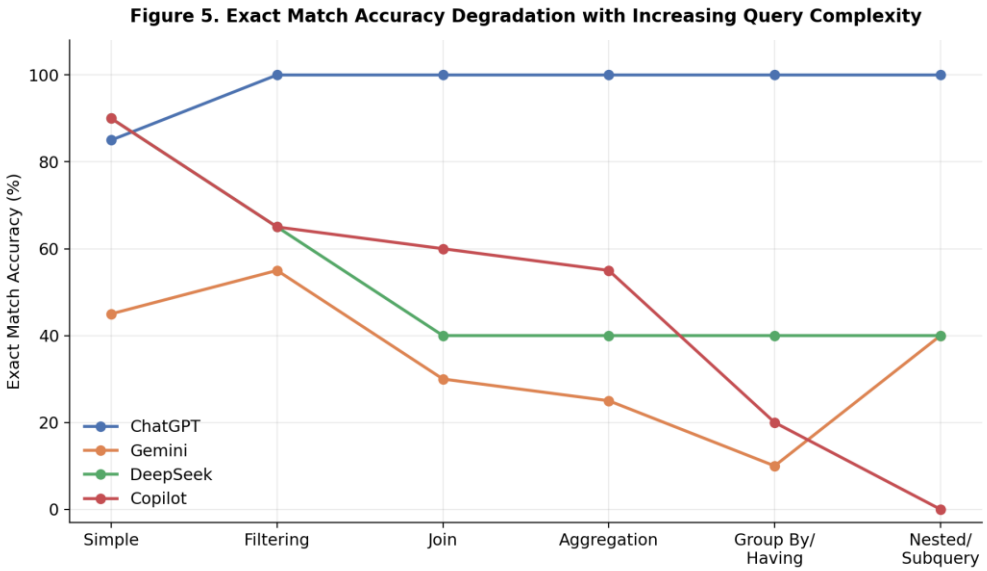

Figure 6. Exact Match Accuracy degradation with increasing query complexity.

4.6 Failure-Mode Analysis

Table 2 illustrates the categorisation of failure in a purpose of Objective 4 (result-set). All 4 models resulted in 27 failures, which were all semantic failures (queries ran successfully, but returned a result set that was different from the ground truth). The number of failures in each category are shown in Table 6, by model. The column totals are assigned to categories—this means that the data determines the column totals, and weaknesses can be spotted by assigning categories to the column totals, for example, weaknesses in the join column, such as join failures caused by join conditions, or weaknesses in the join, such as join failures caused by grouping or nested-logic errors.

Table 6

Execution (Result-Set) Failures by Model and Failure Category (counts)

Failure Category	ChatGPT	Gemini	DeepSeek	Copilot	Total
Schema-linking error	3	0	0	0	3
Join error	0	13	3	0	16
Aggregation/grouping error	0	0	0	0	0
Nested-logic error	0	0	4	4	8
Semantic misinterpretation	0	0	0	0	0
Total failures	3	13	7	4	27

Of all the errors, it was only schema-linking errors that it often retrieved the wrong column or table when the question was for a specific column or table, and all of these errors were of type simple-retrieval; it was a 100% correct on the errors of joining, grouping and nesting. The poor execution accuracy (71.4%) and result-set errors (13/13) of Gemini were due to conditions that wrongly joined the tables or that the tables were joined the wrong way. The result can be seen as a consequence of the fact that result-set errors were entirely due to the wrong join conditions or types, and not to aggregation or nesting. DeepSeek had seven mistakes, all of which were either join errors (three) or nested-logic errors (four), suggesting problems with multi-table relational reasoning and problems with subqueries execution. The four errors made by the Microsoft Copilot were all nested-logic errors, and most of the reliability issues for the less complex constructs. There were no errors relating to aggregation/grouping or semantic misinterpretation (executable SQL that answers a different question) that would suggest that, for this benchmark, errors in results were mainly related to relational structure (joins and nesting) issues.

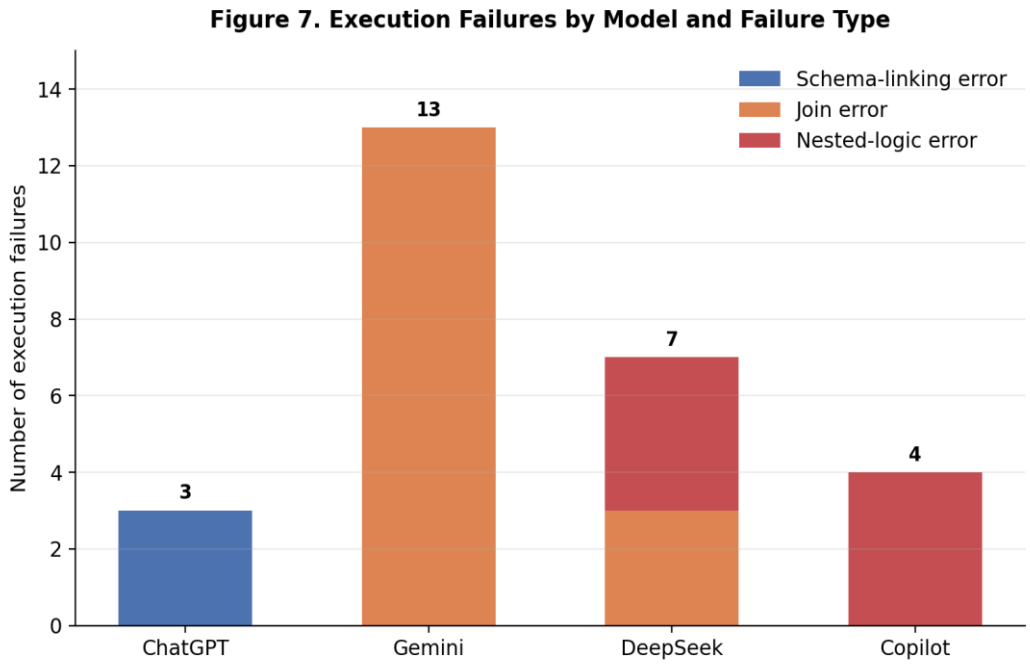

Figure 7. Execution failures by model and failure type.

4.7 Summary of Results

ChatGPT had the best overall scores on both metrics and on all categories. Microsoft Copilot was able to compete with ChatGPT's execution accuracy, DeepSeek had good execution accuracy and an average exact-match; and Gemini had subpar overall performance, but in most categories was still relatively good. The performance for each model decreased with increasing complexity of the query; the filtering queries were the easiest to handle. The main empirical finding of the study is that the EMA–EXA divergence is large and consistent in three of the four models.

5. Discussion

In this study, four modern LLMs are explored in terms of Exact Match and Execution Accuracy on the Text-to-SQL task. The results indicated significant differences between the models and more significantly, persistent semantic/syntactic errors which varied across the models. But ChatGPT topped both counts with 97%/97%, while Microsoft Copilot had a respectable exact match score of 56% and execution accuracy of 96%, and Deepseek was 55%/93%, and Gemini 36%/87%. The patterns give a sense of the state of the art with regards to Text-to-SQL with LLMs, and how Text-to-SQL systems ought to be judged.

The results outlined are in line with the theory of syntactic and semantic equivalence, but it is not.The results are theoretically consonant with the syntactic/semantic distinction, but such is not the case. Evaluation frameworks based on Exact Match assume that a generated query is similar to a reference query, but relational databases can support multiple queries which will return the same results. The syntactic differences between EMA and EXA are evident in the wide gaps between the three models, with Copilot, DeepSeek and, notably, Gemini being more separated from EXA than the other models. This is because the user is more interested in getting the correct answer than in repeating a specific query structure, and execution-based evaluation is more representative of the true effectiveness of the models.

The capacity to perform well is comparable to the other GPT models that appeared to be outstanding at reasoning and understanding contexts[22] and our results apply to an enterprise-style schema, not a standard schema. One of these results is noteworthy and should be analyzed: ChatGPT's gap between the two scores was 0 points, while the other models scored between 38 and 51 points. This could be because the manual queries that were used for reference had similar stylistic elements (formatting, aliasing, ordering of clauses) to the more common style of answers generated by ChatGPT, with some of the similarity attributed to stylistic similarity and not content similarity. However,

this shouldn't detract from ChatGPT's impressive execution performance, and it should also be noted that a match score is not directly comparable with other models, and it should support the core thesis on the importance of execution-based assessments in the paper.

The case of Microsoft Copilot is especially revealing, with 56% accuracy when it comes to accuracy and 96% execution accuracy, including a 0% accuracy when it comes to the accuracy of a nested subquery, but an execution accuracy of 90%. This is a clear example of how exactly match might be underestimating ability for a model that would like to use other, but still sound, formulations. DeepSeek's distribution pattern was similar but not as severe (55%/93%), indicating that the model could understand the semantic meaning through non-reference structures, but there was still some difficulty with the more complex structures. It was the worst performer overall, but Gemini did a relatively good job in most of the categories, and it follows the general pattern that it can be challenging to get good results in the schema linking and relational reasoning categories for some models. These results are corroborated by the failure-mode analysis (Section 4.6): Gemini's errors were all join errors; DeepSeek had a split between join and nested-logic errors; Copilot's errors were all nested-logic errors; and ChatGPT's few errors were caused by schema-linking mistakes on simple queries. That is, the models didn't fail because they failed to understand the questions or to handle aggregation, they failed because they failed to understand the relational structure (the joins and the nesting logic, which is correct) - and that's where SQL complexity lies.

The results confirm a known pattern: filtering and simple retrieval are most easily done, joins, GROUP BY and HAVING clauses and nested subqueries are the most difficult, and need to be operated on correctly and with multi-step reasoning, which is complex given relational dependencies. It seems that while the output might not be identical, existing LLM models generally can generate functional SQL for more complex tasks, as evidenced by the fact that that didn't drop as much as exact-match.

The methodological aspect of the study is its main contribution. Weighting by Exact Match (which is a part of evaluation systems) is likely to lead to misjudging of practical ability. The data here aligns with the change to execution-aware frameworks and to the characterization of the disagreement between the metrics instead of singing a single winner: Execution Accuracy is a better measure of task success, while Exact Match can still signal when a query happens to return correct answers for a small sample of data. Reporting both and analysing the differences between them gives a truer picture than reporting either one or the other. Another observation is that there was no syntax/dialect issue for all four models, and that they all produced executable “Oracle SQL for all the questions. All the errors were semantic - that is, queries that run without error and return the wrong rows - so any surface level validity checking would have been deemed correct, and only the result level evaluation would have spotted the true errors.

In practice, the results show that modern LLMs like ChatGPT and Microsoft Copilot can support many tasks that involve queries against a database and can diminish technical obstacles to accessing data. However, due to the range of performance levels in the complex categories there is a potential risk in using unlimited use in high stakes situations. Organisations should thus complement LLM-based questioning with checks, validity, and human supervision of the information retrieved, and governance bodies might require guidelines on how to validate the information retrieved by LLM before taking any crucial decisions.

There are a number of caveats to be noted. First, this benchmark was constructed using a 100 question set based on a 5-table schema, but it does not span complexity of enterprise databases, nor does it cover the range and size of enterprise databases. Secondly, there are few questions in the group by/having category and the few questions in the nested subquery category have a single most noticeable value (with a 0% exact-match on the nested queries) that will be based on 10 items. Third, the results of only four models were tested: others (such as Claude or domain-specific systems that are part of the Llama family) might perform differently. Fourth, the question texts in natural language were directly presented to the models without any schema-directed prompts.Fourth, the free-form question texts were input to the models with no schema-injected prompts. This was done in order to simulate the "real-world" use of this system, and the same set of questions was presented to all models, but the prompt wording is not controlled and so any differences noted could be attributed to differences in model performance, or attributed to differences in how the models responded to the conversational input, which under different prompt configurations might be different. Fifth, as these are changing commercial products, future updates could impact upon their reproducibility and therefore the exact version and access date have been noted. Sixth, "exact-match" was strict on raw string matching - a lower bound for syntactic agreement and further demarks the results of the two measures. Lastly, since Execution Accuracy was tested with just one populated instance, a generated query could (in principle) return the same rows as the ground truth by chance on this data set, but be different on other data sets, so a multiple test set or multiple test databases are future motivators.

Future research should examine larger and more complicated schemas and multiple benchmarks, as well as other models, and additionally examine the effect of prompt-engineering and retrieval-augmented approaches, which provide dynamic schema context. The introduction of measures of semantic-equivalence other than the execution check on a single dataset would be an additional aid to evaluation, as would the implementation of studies of successive versions of the models over a period of time.

6. CONCLUSION

This paper benchmarked and contrasted the performance of ChatGPT, Google Gemini, DeepSeek, and Microsoft Copilot in generating natural language-to-SQL queries on an enterprise-like schema and evaluated them under six complexity categories, using the two metrics: Execution Accuracy and Exact Match. ChatGPT gave the best performance (97% on both measures). Microsoft Copilot: 56% and 96% respectively; DeepSeek: 55% and 93% respectively; and Gemini: 36% and 87% respectively. The filtering of queries was best supported, joins and group-by/having were least supported, and nested subqueries were least supported. The most significant result was the large and quite consistent difference between results of Exact Match and Execution accuracy for three out of four models, which revealed that syntactic similarity is not enough to judge the Text-to-SQL systems.

The overall impact is not only practical, but also methodological. Evidence suggests that these metrics should be used to evaluate products with execution in mind and that it is insufficient to rely on rankings of products that are identical (Exact Match). In practical terms, the impressive execution precision of ChatGPT and Microsoft Copilot suggests significant potential to democratize access to and speed of analytics,with appropriate validation and human oversight required during implementation—especially for complex queries. In the current scope—enough: a small benchmark, four models, one prompting strategy, and reliance on the Exact Match normalization used—the study enhances the comparative understanding of modern LLM and provides a template that can be easily implemented for execution-aware Text-to-SQL evaluation.

Declarations

Funding. The authors received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors for this work.

Conflicts of Interest. The authors declare that they have no competing interests relevant to the content of this article.

Ethical Approval. This study did not involve human participants or animals; formal ethical approval was therefore not required, as detailed in Section 3.2.

Data Availability. The benchmark questions, ground-truth SQL queries, and raw model outputs supporting the findings of this study are available from the corresponding author on reasonable request.

Use of AI Tools. The four large language models examined in this study (ChatGPT, Google Gemini, DeepSeek, and Microsoft Copilot) are the objects of evaluation and are described in the Methods. Separately, the authors used an AI assistant (Claude, Anthropic) to assist with language editing, structuring, and formatting of the manuscript. The authors performed and verified all study design, data collection, analysis, interpretation, and final decisions about the content, and they take full responsibility for the integrity and accuracy of the work.

Appendix A. Schema-Generation Prompt

The five-table evaluation schema was generated with the assistance of a large language model using the single design prompt reproduced below, after which the authors reviewed, corrected, and finalised the resulting CREATE TABLE statements and sample data.

Act as a database architect. Create a realistic enterprise relational database schema for evaluating Natural Language to SQL (Text-to-SQL) systems.

Requirements:

1. Create five related tables: CUSTOMERS, PRODUCTS, ORDERS, ORDER_ITEMS, EMPLOYEES.

2. Define: Primary Keys; Foreign Keys; Appropriate data types.

3. Ensure the schema supports: Simple SELECT queries; Filtering queries; JOIN operations; Aggregation functions; GROUP BY and HAVING clauses; Nested and Subquery operations.

4. Provide: CREATE TABLE statements; Sample INSERT statements; At least 5–10 records per table.

5. The schema should resemble a real-world retail or enterprise sales environment suitable for benchmarking Large Language Models for Text-to-SQL generation.

Appendix B. Benchmark of 100 Natural Language Questions

The complete set of 100 natural language questions, grouped by SQL complexity category, is listed below. The same questions, in the same order, were posed to all four models on 12 June 2026.

Simple Retrieval (Q1–Q10)

Q1. Show all customers.

Q2. Show all products.

Q3. Show all orders.

Q4. Show all employees.

Q5. List all customer names.

Q6. List all product names.

Q7. Show all product categories.

Q8. Show all employee departments.

Q9. Show all order dates.

Q10. List all cities where customers are located.

Filtering (Q11–Q40)

Q11. Show all customers from Bangalore.

Q12. Show all customers from Mumbai.

Q13. List products in Electronics category.

Q14. List products in Furniture category.

Q15. Show employees working in IT department.

Q16. Show employees working in Sales department.

Q17. Show products costing more than 10000.

Q18. Show products costing less than 10000.

Q19. Show orders placed after 01-JAN-2024.

Q20. Show customers registered after 01-MAR-2023.

Q21. Show customers from Karnataka.

Q22. Show customers from Telangana.

Q23. Show products priced between 5000 and 30000.

Q24. Show products costing more than 25000.

Q25. Show employees earning more than 60000.

Q26. Show employees earning less than 50000.

Q27. Show orders with total amount greater than 50000.

Q28. Show orders with total amount less than 10000.

Q29. Show customers whose names start with 'A'.

Q30. Show products whose names contain 'Desk'.

Q31. Show employees in IT earning more than 70000.

Q32. Show Electronics products costing above 20000.

Q33. Show orders placed in February 2024.

Q34. Show customers registered during 2023.

Q35. Show employees whose manager_id is not null.

Q36. Show customers from Bangalore and Karnataka.

Q37. Show Furniture products costing less than 15000.

Q38. Show orders between January and March 2024.

Q39. Show employees with salaries between 45000 and 70000.

Q40. Show customers whose names end with 'a'.

Join (Q41–Q60)

Q41. Show customer names and their order dates.

Q42. Show customer names and order amounts.

Q43. Show customer names and cities along with order amounts.

Q44. Show products purchased in each order.

Q45. Show order details with customer names.

Q46. Show product names and quantities ordered.

Q47. Show customers and products they purchased.

Q48. Show customer names and total number of orders.

Q49. Show products purchased by Rahul Sharma.

Q50. Show products purchased by customers from Bangalore.

Q51. Show order amounts for customers from Mumbai.

Q52. Show employees and their managers.

Q53. Show manager names for each employee.

Q54. Show customer names and purchased product names.

Q55. Show all order items with product names.

Q56. Show customer cities and purchased product categories.

Q57. Show products sold in order 1001.

Q58. Show the names of customers who purchased Electronics products.

Q59. Show the names of customers who purchased Furniture products.

Q60. Show employee names and their department managers.

Aggregation (Q61–Q80)

Q61. Show total sales amount.

Q62. Show average order amount.

Q63. Show maximum order amount.

Q64. Show minimum order amount.

Q65. Count total customers.

Q66. Count total products.

Q67. Count total employees.

Q68. Count orders placed in 2024.

Q69. Show total sales by city.

Q70. Show total sales by product category.

Q71. Show average salary by department.

Q72. Show total number of orders per customer.

Q73. Show total quantity sold per product.

Q74. Show highest priced product in each category.

Q75. Show lowest priced product in each category.

Q76. Show total revenue generated by each product.

Q77. Show number of customers in each city.

Q78. Show number of products in each category.

Q79. Show total salary expense by department.

Q80. Show average product price by category.

Group By / Having (Q81–Q90)

Q81. Show cities having more than one customer.

Q82. Show customers who placed more than one order.

Q83. Show departments having more than one employee.

Q84. Show product categories having more than one product.

Q85. Show customers whose total purchases exceed 50000.

Q86. Show products sold more than twice.

Q87. Show cities with total sales greater than 50000.

Q88. Show departments whose average salary exceeds 60000.

Q89. Show categories generating revenue above 20000.

Q90. Show customers with more than two purchased products.

Nested / Subquery (Q91–Q100)

Q91. Show employees earning above average salary.

Q92. Show customers whose purchase amount is above average order amount.

Q93. Show products priced above average product price.

Q94. Show customers who placed the highest-value order.

Q95. Show employees earning the maximum salary.

Q96. Show products never purchased.

Q97. Show customers who never placed an order.

Q98. Show products belonging to the most expensive category.

Q99. Show employees working in the department with the highest average salary.

Q100. Show customers whose total purchases are greater than the average customer purchase amount.

References

1. J. Liu et al. , “A Survey of Text-to-SQL in the Era of LLMs: Where Are We, and Where Are We Going?,” IEEE Transactions on Knowledge and Data Engineering , vol. 37, no. 10, pp. 5735–5754, Jul. 2025, doi: 10.1109/tkde.2025.3592032.
2. M. Liu, X. Wang, J. Xu, W. Yi, and O. Wolfson, “A systematic review of natural language interfaces for databases,” Frontiers of Computer Science , vol. 20, no. 11, Mar. 2026, doi: 10.1007/s11704-025-50592-w.
3. G. Katsogiannis-Meimarakis and G. Koutrika, “A survey on deep learning approaches for text-to-SQL,” The VLDB Journal , vol. 32, no. 4, pp. 905–936, Jan. 2023, doi: 10.1007/s00778-022-00776-8.
4. A. Rahman et al. , “Comparative analysis based on DeepSeek, ChatGPT, and Google Gemini: Features, techniques, performance, future prospects,” Systems and Soft Computing , vol. 7, pp. 200396–200396, Oct. 2025, doi: 10.1016/j.sasc.2025.200396.
5. C. Shorten et al. , “Querying Databases with Function Calling,” Jan. 23, 2025. doi: 10.48550/arxiv.2502.00032.
6. W. Zhang et al. , “Natural Language Interfaces for Tabular Data Querying and Visualization: A Survey,” IEEE Transactions on Knowledge and Data Engineering , vol. 36, no. 11, pp. 6699–6718, May 2024, doi: 10.1109/tkde.2024.3400824.
7. Y. Kong, H. Hu, D. Zhang, S. Chai, F. Zhang, and W. Wang, “Bridging the Gap: Transforming Natural Language Questions into SQL Queries via Abstract Query Pattern and Contextual Schema Markup,” Feb. 20, 2025. doi: 10.48550/arxiv.2502.14682.
8. L. Shi, Z. Tang, N. Zhang, X. Zhang, and Z. Yang, “A Survey on Employing Large Language Models for Text-to-SQL Tasks,” Jun. 03, 2025, Cornell University . doi: 10.1145/3737873.
9. L. Ma, K. Q. Pu, and Y. Zhu, “Evaluating LLMs for Text-to-SQL Generation With Complex SQL Workload,” Jul. 28, 2024, Cornell University . doi: 10.48550/arxiv.2407.19517.
10. B. Li, Y. Luo, C. Chai, G. Li, and N. Tang, “The Dawn of Natural Language to SQL: Are We Fully Ready?,” Proceedings of the VLDB Endowment , vol. 17, no. 11, pp. 3318–3331, Jul. 2024, doi: 10.14778/3681954.3682003.
11. B. G. Ascoli, Y. S. R. Kandikonda, and J. D. Choi, “ETM: Modern Insights into Perspective on Text-to-SQL Evaluation in the Age of Large Language Models,” Future Internet , vol. 17, no. 8, pp. 325–325, Jul. 2025, doi: 10.3390/fi17080325.
12. C. Renggli, I. F. Ilyas, and T. Rekatsinas, “Fundamental Challenges in Evaluating Text2SQL Solutions and Detecting Their Limitations,” Jan. 30, 2025. doi: 10.48550/arxiv.2501.18197.
13. C. B. Dadi, “Natural Language Interfaces for Database Management: Bridging the Gap Between Users and Data through Conversational AI,” Journal of Computer Science and Technology Studies , vol. 7, no. 3, pp. 927–933, May 2025, doi: 10.32996/jcsts.2025.7.3.103.
14. S. Tata and G. M. Lohman, “SQAK,” pp. 889–902, Jun. 2008, doi: 10.1145/1376616.1376705.
15. H. V. Jagadish et al. , “Making database systems usable,” pp. 13–24, Jun. 2007, doi: 10.1145/1247480.1247483.
16. Z. Hong et al. , “Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL,” Jun. 12, 2024, Cornell University . doi: 10.48550/arxiv.2406.08426.
17. J. Li et al. , “Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs,” May 04, 2023, Cornell University . doi: 10.48550/arxiv.2305.03111.
18. A. B. Kanburoğlu and F. B. Tek, “Text-to-SQL: A methodical review of challenges and models,” TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES , vol. 32, no. 3, pp. 403–419, May 2024, doi: 10.55730/1300-0632.4077.
19. E. Bethel et al. , “Case Study: Leveraging GenAI to Build AI-based Surrogates and Regressors for Modeling Radio Frequency Heating in Fusion Energy Science,” arXiv (Cornell University) , Sep. 2024, doi: 10.48550/arxiv.2409.06122.
20. A. Grange, “Learning SQL from within: integrating database exercises into the database itself,” Oct. 21, 2024, Cornell University . doi: 10.48550/arxiv.2410.16120.
21. B. Qin et al. , “A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions,” Aug. 29, 2022, Cornell University . doi: 10.48550/arxiv.2208.13629.
22. H. Zhang, R. Cao, L. Chen, H. Xu, and K. Yu, “ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought,” pp. 3501–3532, Jan. 2023, doi: 10.18653/v1/2023.findings-emnlp.227.