Computer Science

Khoroshevskyi Mykola
Principal software engineer - Sigma Software
(Altea, Spain)

BUILDING PHYSICAL DATA ACCESS PLANS USING LARGE LANGUAGE MODELS

Summary. *The theoretical and practical foundations of designing physical data access plans using large language models are explored. The structure of a physical plan, main data access techniques, and connection algorithms are explained. The potential of language models for transforming SQL queries, selecting control instructions, and generating execution options is revealed. A sequence for creating a physical plan is provided, which involves analyzing the original query, selecting a transformation rule, generating a new query, verifying its syntax and semantics, creating a plan with a database management system, and evaluating performance. It is determined that using language models necessitates mandatory verification of query equivalence, operation validity, and actual execution time. Conditions under which the expense of using the model may be offset by increased data processing efficiency are identified.*

Key words: *large language model, physical plan, data access, query optimization, SQL query, database management system, connection algorithms, query rewriting, cost estimation, execution time.*

Relevance of the study. The increase in data volumes and the complexity of information systems place new demands on the performance of database management systems. Query processing speed is largely determined by the physical data access plan, which sets the order of scanning tables and indexes, connecting data, and computing operations. Choosing the optimal plan is

becoming an increasingly difficult task, as there are many possible options and estimating the cost of each depends on the accuracy of statistical data.

The development of large language models opens up new horizons for the analysis of SQL queries, database structures, and execution plans [9]. These models can be used to create plans, select physical operations, and compare different data access options. However, the solutions they offer need to be validated and evaluated using database management system (DBMS) tools. Therefore, the study of methods for constructing physical data access plans based on large language models is an urgent area that can complement traditional query optimization methods.

The purpose of the study. The aim of this research is to explore the potential and process of applying large language models to the creation and optimization of data access strategies for physical systems, as well as to establish criteria for verifying the accuracy and efficacy of the solutions generated.

Materials and research methods. The research materials included scientific publications on query optimization, physical plan construction, and the use of large language models in database management systems. Additionally, open technical documentation was used.

The paper employs methods of analysis and synthesis of scientific literature, comparative analysis of plan-building techniques, systematization of physical operations, and a structural description of the steps in the transformation of SQL queries.

The results of the study. A physical plan is a tree-like sequence of operations that a database management system performs to obtain a query result. The end nodes of the tree access tables or indexes, and the higher nodes perform filtering, joining, sorting, grouping, and calculating totals. Each node accepts strings from child operations, processes them, and passes the result to the next node. The root node forms the final data set [1].

An example of the structure of the physical query execution plan is shown in Figure 1.

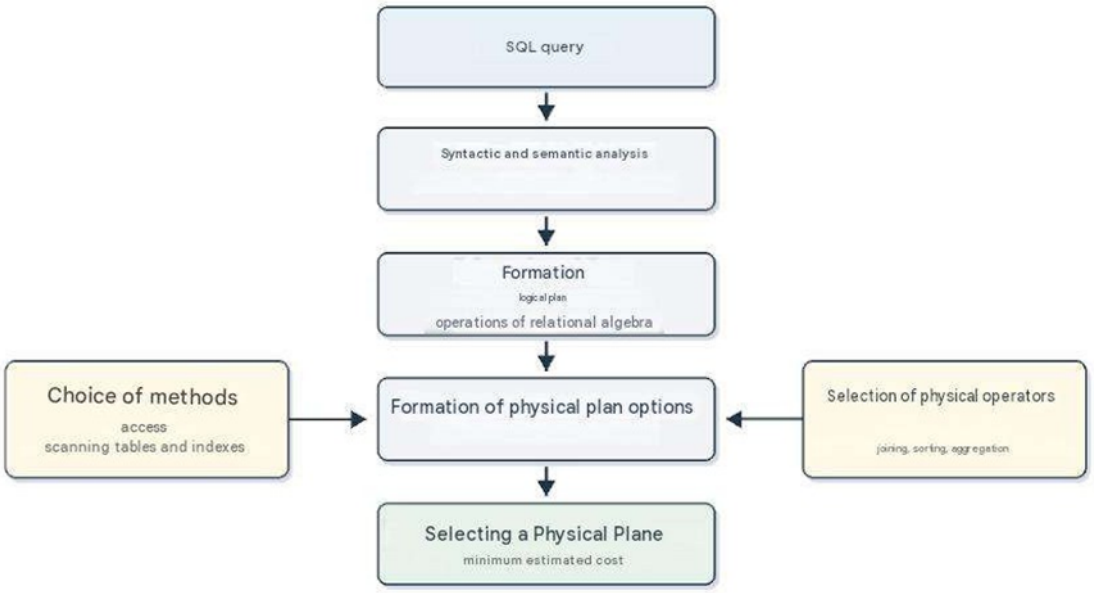

Fig. 1. An example of the structure of the physical query execution plan

Source: author's development

The process of building a physical plan begins with analyzing the query at the syntactic and semantic levels. During this analysis, the system determines tables, columns, selection conditions, and the relationships between the various entities. Then, it considers possible ways to perform the operations. The same query can be executed using different methods, and the task of the scheduler is to select the option with the lowest expected cost.

There are several basic methods for accessing data. Sequential scanning involves reading the entire table in order. An index scan uses an index to find rows that match a given condition. If the required columns are included in the index and the conditions for its use are met, index-only scanning can retrieve the necessary values without having to access the main table. During bitmap scanning, a list of suitable row locations is generated first, and then the corresponding table pages are accessed.

When processing queries involving multiple tables, the order and method for joining them must be chosen (Table 1) [3].

Table 1

Basic data connection algorithms

Algorithm	Necessary condition	Main Feature
Nested loops	The possibility of multiple searches in the internal set	The internal operation is performed for the rows of the external set
Merge	Data ordering by connection keys	Both sets are processed sequentially
Hash connection	The ability to apply hashing to the key	A hash table is pre-built for one set

Source: author's development

The choice of operations is based on an estimate of the number of rows to be processed at each stage. Statistical data, such as the number of rows, the distribution of values, the proportion of empty values, and the most common values, are used to calculate this estimate [4]. An inaccurate assessment can lead to an incorrect connection and the choice of an inappropriate access method. For example, if there is a relationship between two columns, separate evaluation of conditions may show a single suitable row instead of hundreds of actual rows. Multidimensional statistics can take into account these dependencies and improve the accuracy of calculations, ensuring a more precise estimation.

The cost of a plan is an estimated amount that includes estimated costs for page reading, row processing, sorting, and other operations. For each node, the initial cost, the total cost, the expected number of rows and the average row size are determined. The cost of the upstream node includes the costs of the child operations. After comparing the acceptable options, the plan with the lowest estimated cost is selected. However, for a final assessment of its quality, it is necessary to compare the projected and actual performance indicators [7].

Large language models are used at various stages of query optimization: they can change the text of SQL queries, select optimal control instructions for the optimizer, compare ready-made plans, and directly generate a physical plan. The input data for the model is the query itself, a description of the database schema, information about indexes, statistical data, and examples of previously completed plans. As a result of the model's operation, a transformed query, a set of guidelines for a database management system, or a structured representation of a new plan can be created.

One of the ways to use LLM is to rewrite an SQL query while preserving its original meaning. The model may suggest moving the filtering conditions, changing the form of the nested query, or suggesting another acceptable way to write an expression. However, despite the changes in the text, a new physical plan does not always appear. Therefore, DBMS tools [10] should check the correctness of the conversion and the actual execution time.

Another approach is to use a model to assist an existing optimizer. In this approach, the LLM does not generate a complete plan but rather selects hints that influence the order of table joins and individual operations. The LLM-based system presents plans as vectors, identifies the most similar plans among previously executed queries, and generates suggestions based on these examples.

A large language model can also generate a plan directly. In the LLM-QO system, queries, metadata, and plans are converted to text format. Training is conducted in two stages: first, the model learns how to generate acceptable plans based on examples, and then it gives preference to options with better performance indicators. Experiments have shown that it is possible to generate plans for queries whose structure was not found in the training set. However, the authors also noted cases where unacceptable plans were generated during additional training, which confirms the need for automatic verification of results [2].

The language model may create a syntactically incorrect plan, apply an operation that is not supported, or suggest an option that will take longer to execute than the original one. Therefore, before using LLM in practice, it is necessary to check semantic equivalence, operator validity, and actual performance.

Thus, large language models can perform three main functions: to suggest optimization options, guide the search using hints, and improve existing plans. The most controlled approach is one in which the model offers solutions, and the database management system verifies them and selects the final option based on the results.

One of the ways to use powerful language models to create physical plans is to automatically convert queries into Structured Query Language (SQL). The language model does not generate the physical plan directly but creates an equivalent version of the original query. Then, the built-in database management system optimizer creates a new physical plan for the modified query [6].

At the first stage, the system receives the initial request, information about the database structure, and a physical plan for its execution. This data allows you to determine which parts of the request may slow down the processing process. The system then offers a text recommendation describing a possible way to convert the query.

This system uses rewriting rules formulated in natural language. Each rule contains the conditions under which the transformation should be applied and a description of the necessary changes. Successfully applied rules are saved and can be reused to optimize queries with a similar structure. This allows the system to apply the results of previously performed optimization when solving new tasks.

Based on the selected rule, the large language model generates a new SQL query. The received query is first checked for syntax errors. Then, the semantic equivalence of the original and modified queries is checked; they should produce the same results with the same source data [5].

If the check finds a difference, the system generates an example of where the query results do not match. This example is then passed to the language model, along with an error description. The model then corrects the request, and the verification process is repeated. This process allows for consistent elimination of syntactic and semantic errors.

Figure 2 shows the sequence of building a physical plan using a large language model.

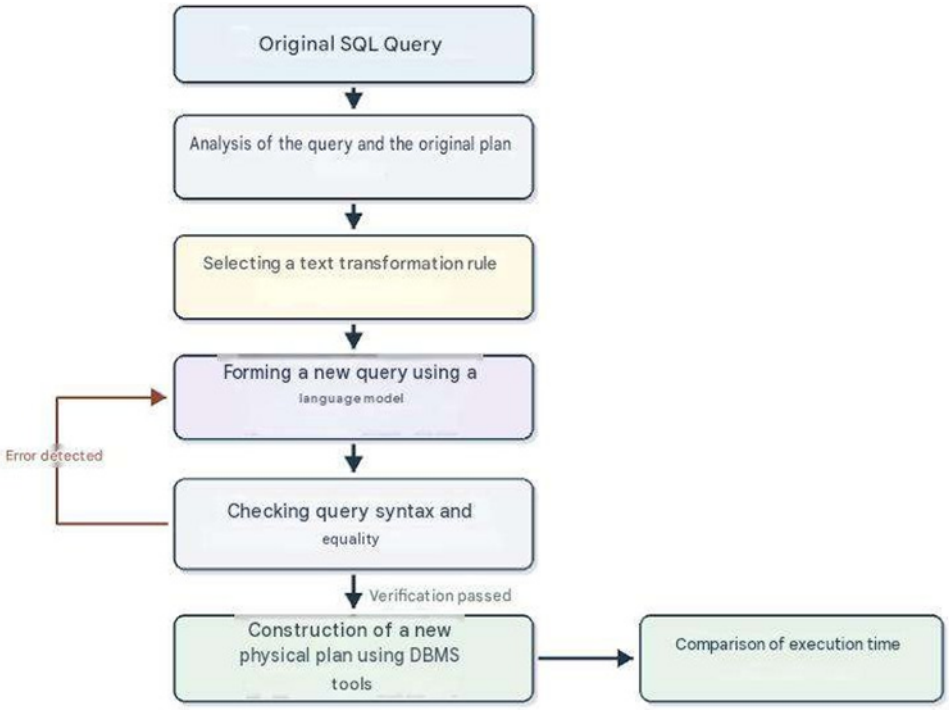

Fig. 2. The sequence of building a physical plan using a large language model

Source: author's development

For each verified query, the built-in database management system (DBMS) optimizer creates a new physical plan. After that, the original and transformed queries are executed under the same conditions. An option is considered effective if it gives the correct result and reduces data processing time.

The main stages of the formation of the physical plan are presented in Table 2.

Table 2

The main stages of building a physical plan

Stage	Initial data	Result
Analysis	SQL query, database structure and initial plan	Determining the direction of transformation
Choosing a rule	Saved text recommendations	The appropriate way to change the request
Transformation	Request and selected rule	New SQL Query
Syntax checking	Modified request	Confirmation of the correct entry
Checking the equivalence	Original and modified queries	Confirmation of matching results
Correction	Error description and example of discrepancy	Updated version of the request
Building a plan	Verified request	The new physical plane
Evaluation	Original and new plans	Choosing a faster option

Source: author's development

The quality of a method should not be evaluated solely by the time it takes to complete a single query. It is essential to consider the costs of accessing a language model, the time required to create a plan, memory usage, and the consistency of the results when the amount of data changes. A fast plan based on a small dataset may lose its effectiveness after expanding it or altering the distribution of values.

To ensure a fair comparison, the experiment should involve repeated execution of each option under identical conditions. Both the average time and the delay for the 90th percentile should be taken into account, reflecting the performance of the slowest queries.

It is also necessary to check how the method works with new, previously unknown queries. If the training and verification data have the same structure, then the high result may be due to memorizing patterns. Therefore, queries should be divided into independent groups, and the transformed variants should be additionally checked after changing statistics and available indexes.

The practical application of this method is advisable, first of all, for repetitive and resource-intensive queries. In this case, the cost of analyzing several options is offset by their subsequent repeated execution. However, for short or one-time queries, the cost of accessing the model and additional verification may exceed the resulting time reduction [8].

Conclusions. Thus, large language models can be useful for reformulating SQL queries, selecting control instructions, and creating alternative physical plans. However, you should not consider language models as a full-fledged replacement for the built-in optimizer. A collaborative approach is more effective, in which the language model offers conversion options, and the database management system verifies their correctness, builds an executable plan, and evaluates actual performance. The effectiveness of this method depends on the accuracy of the source data, the quality of the proposed solutions, and the query execution conditions. It is especially useful to use this approach for repetitive and resource-intensive operations. In such cases, the cost of running the model and checking multiple options can be offset by a subsequent reduction in data processing time.

References

1. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., & Neumann, T. (2015). How good are query optimizers, really? *Proceedings of the VLDB Endowment, 9*(3), 204–215.

2. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 30*, 5998–6008.

3. Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J., & Stoica, I. (2018). *Learning to optimize join queries with deep reinforcement learning*.

4. Kipf, A., Kipf, T., Radke, B., Leis, V., Boncz, P. A., & Kemper, A. (2019). Learned cardinalities: Estimating correlated joins with deep learning.

Proceedings of the 9th Biennial Conference on Innovative Data Systems Research.

5. Marcus, R., Negi, P., Mao, H., et al. (2019). Neo: A learned query optimizer. *Proceedings of the VLDB Endowment, 12*(11), 1705–1718.

6. Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

7. Negi, P., Marcus, R. C., Kipf, A., et al. (2021). Flow-loss: Learning cardinality estimates that matter. *Proceedings of the VLDB Endowment, 14*(11), 2019–2032.

8. Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., & Kraska, T. (2021). Bao: Making learned query optimization practical. *Proceedings of the 2021 International Conference on Management of Data*, 1275–1288.

9. Ouyang, L., Wu, J., Jiang, X., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems, 35*, 27730–27744.

10. Yang, Z., Liang, E., Kamsetty, A., et al. (2022). Balsa: Learning a query optimizer without expert demonstrations. *Proceedings of the 2022 International Conference on Management of Data*, 931–944.