LANTERN: Boredom-conscious Natural Language Description Generation of Query Execution Plans for Database Education

Peng Chen
Xidian University
China
pchen97@stu.xidian.edu.cn

Hui Li
Xidian University
China
hli@xidian.edu.cn

Sourav S Bhowmick
Nanyang Technological University
Singapore
assourav@ntu.edu.sg

Shafiq R Joty
Nanyang Technological University
Singapore
srjoty@ntu.edu.sg

Weiguo Wang
Xidian University
China
wgwang@stu.xidian.edu.cn

ABSTRACT

The database systems course in an undergraduate computer science degree program is gaining increasing importance due to the continuous supply of database-related jobs as well as the rise of Data Science. A key learning goal of learners taking such a course is to understand how SQL queries are executed in an RDBMS in practice. An RDBMS typically exposes a *query execution plan* (QEP) in a visual or textual format, which describes the execution steps for a given query. However, it is often daunting for a learner to comprehend these QEPs containing vendor-specific implementation details. In this demonstration, we present a novel, generic, and portable system called LANTERN that generates a natural language (NL)-based description of the execution strategy chosen by the underlying RDBMS to process a query. It provides a *declarative framework* called POOL for subject matter experts (SME) to efficiently create and manipulate the NL descriptions of physical operators of any RDBMS. It then exploits POOL to generate the NL descriptions of QEPs by *integrating* a *rule-based* and a *deep learning-based* techniques to infuse language variability in the descriptions. Such an NL generation strategy mitigates the impact of *boredom* on learners caused by repeated exposure of similar text generated by a rule-based system.

CCS CONCEPTS

• **Information systems → Database query processing**.

KEYWORDS

database education, query execution plan, natural language generation

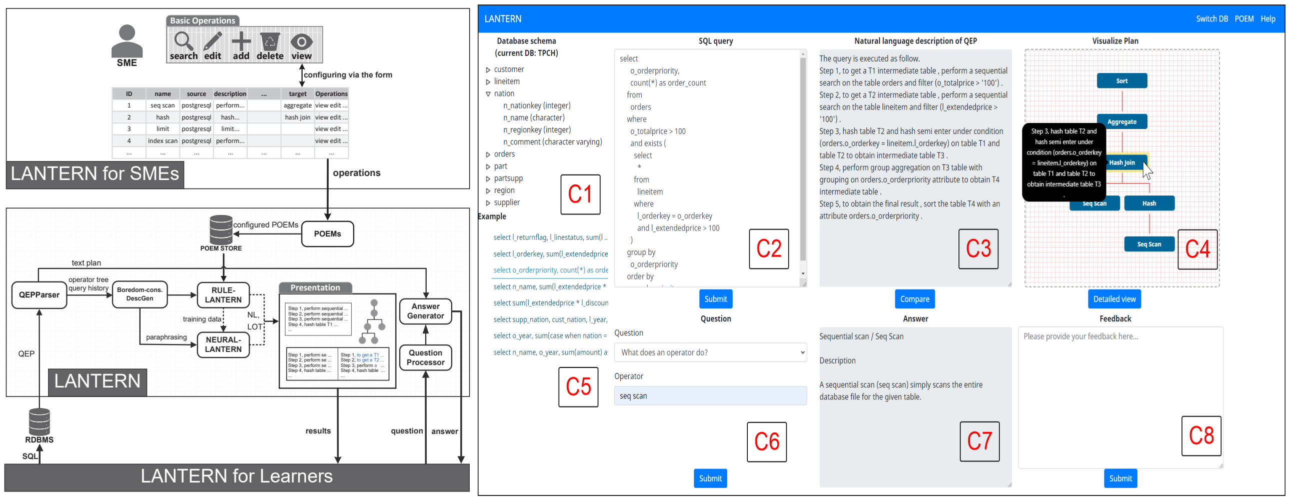
ACM Reference Format:
Peng Chen, Hui Li, Sourav S Bhowmick, Shafiq R Joty, and Weiguo Wang. 2022. LANTERN: Boredom-conscious Natural Language Description Generation of Query Execution Plans for Database Education. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3514221.3520165

1 INTRODUCTION

"Database education is at an inflection point" [8]. Due to the widespread use of relational database management system (RDBMS) in the commercial world as well as the growth of Data Science as a discipline, there is an increasing demand of database-related courses in academic institutions. Learners from diverse fields aspire to take these courses, even with limited Computer Science backgrounds [8]. Hence, learner-friendly tools that can facilitate learning and understanding of database concepts have the potential to augment the traditional modes of learning (*i.e.,* textbook, lecture). This is evident from recent activities in the data management community such as research [11, 15], panels [8], and workshops (*e.g.,* https://dataedinitiative.github.io/).

One of the key goals for learners taking a database course is to understand the execution strategies of SQL queries in practice by an RDBMS. Given an SQL query, the query engine in an RDBMS produces a *query execution plan* (QEP), which represents the execution strategy of the query. Hence, such an understanding can be gained by perusing the QEPs of queries. Major database textbooks introduce *general* (*i.e.,* not tied to any specific RDBMS) theories and principles associated with QEPs using natural language-based narratives and visual examples. This allows a learner to gain a general understanding of SQL query execution strategies.

Most database courses complement the text book-based learning with hands-on interaction with an off-the-shelf commercial RDBMS (*e.g.,* PostgreSQL) to infuse knowledge about database techniques used in practice. A QEP is exposed in a *visual* or a *textual* (*e.g.,* JSON, XML) format in these systems. Unfortunately, comprehending these formats in practice can be daunting for many learners as they demand a knowledge of vendor-specific implementation details [11, 15]. This problem is further aggravated by the deployment of physical operators with different names and functions by different vendors. In fact, a recent survey with real-world learners reveals that a natural language-based description of a QEP (*i.e.,* textbook style) is highly desirable and complements the visual tree-based format [15]. Hence, *a learner-friendly tool that generates descriptions of QEPs in a natural language (NL) can enable learners to understand the query execution steps of SQL queries in practice.*



select l_returnflag, l_linestatus, sum(l ..
select l_orderkey, sum(l_extendedprice
select o_orderpriority, count(*) as orde
select n_name, sum(l_extendedprice *
select sum(l_extendedprice * l_discoun
select supp_nation, cust_nation, l_year,
select o_year, sum(case when nation =
select n_name, o_year, sum(amount) a

```
select
  o_orderpriority,
  count(*) as order_count
from
  orders
where
  o_totalprice > 100
  and exists (
   select
    *
   from
    lineitem
   where
    l_orderkey = o_orderkey
    and l_extendedprice > 100
  )
group by
  o_orderpriority
order by
```

The query is executed as follow.
Step 1, to get a T1 intermediate table , perform a sequential search on the table orders and filter (o_totalprice > '100') .
Step 2, to get a T2 intermediate table , perform a sequential search on the table lineitem and filter (l_extendedprice > '100') .
Step 3, hash table T2 and hash semi enter under condition (orders.o_orderkey = lineitem.l_orderkey) on table T1 and table T2 to obtain intermediate table T3 .
Step 4, perform group aggregation on T3 table with grouping on orders.o_orderpriority attribute to obtain T4 intermediate table .
Step 5, to obtain the final result , sort the table T4 with an attribute orders.o_orderpriority .

Step 3, hash table T2 and hash semi enter under condition (orders.o_orderkey = lineitem.l_orderkey) on table T1 and table T2 to obtain intermediate table T3 .

Sequential scan / Seq Scan

Description

A sequential scan (seq scan) simply scans the entire database file for the given table.

Figure 1: LANTERN: (a) Architecture (left). (b) GUI for learners (right).

In this demonstration, we present a novel, *generic*, and *portable* system called LANTERN [15] (naturaL lANguage descripTion of quERy plaNs) to support a user-friendly NL-based description of the QEP of an arbitrary SQL query posed on an RDBMS. In particular, given an SQL query, instead of a purely visual or semi-structured format of the QEP, LANTERN generates a *multi-faceted* NL-based description of the steps in the QEP. Under the hood, LANTERN incorporates a *declarative framework* called POOL to empower *subject matter experts* (SMEs) create and manipulate the NL descriptions (*i.e.,* labels) of physical operators, which are the building blocks of QEPs. It implements an NL description generation framework that utilizes POOL to *integrate* a rule-based and a deep learning-based techniques in order to infuse language variability in the descriptions opportunely. This strategy mitigates the impact of *boredom* on learners that may arise due to repetitive statements in different NL descriptions generated by a purely rule-based technique [15].

2 DESIGN PHILOSOPHY

LANTERN's design is based on the following four principles:

(1) Portability and generalizability of LANTERN: The NL generation framework should be *generalizable*. That is, it should be easily deployable on any domain-specific applications (*e.g.,* movies, hospitals). Furthermore, it should be easily portable across different RDBMS (*e.g.,* PostgreSQL, DB2, SQLServer). This will significantly reduce the cost of its deployment in different learning institutes where different application-specific examples and RDBMS software may be used to teach a database systems course.

(2) Rich support for SQL of different complexities: LANTERN should be able to generate an NL description of *any* read-only SQL statements posed by a learner. That is, it should be *orthogonal* to the complexities of SQL queries.

(3) Multi-faceted presentation of NL description: Different learners may wish to view an NL description of a QEP in different modes. Some may simply view it in a text format while others may wish to view it in a visual tree mode where the nodes are annotated with corresponding NL descriptions. Hence, LANTERN should support multi-faceted presentation of an NL description of a QEP to cater to different learners.

(4) Boredom-conscious NL description generation: A rule-based NL description generation framework (*e.g.,* NEURON [11]) may often result in learners feeling boredom[1] after reading descriptions of several queries due to the repeated exposure of same/similar text to describe the QEPs [15]. Some reported that they started skipping several sentences in the descriptions [15]. This is, in fact, consistent with research in psychology that have found that repetition of text messages can lead to annoyance and boredom [4] resulting in purposeful avoidance [6] and content blindness [7]. Hence, LANTERN should be able to inject diversity in the NL descriptions judiciously in order to mitigate the impact of boredom.

3 SYSTEM OVERVIEW

The architecture of LANTERN is outlined in Figure 1(a). Here we briefly describe the key components. The reader may refer to [15] for details.

3.1 The GUI Module

LANTERN exposes two visual interfaces, one for learners and another for *subject matter experts* (SME). The former is for end users who use LANTERN to view the NL descriptions of QEPs. The latter is for database experts who create and manipulate the NL descriptions of various physical operators in different RDBMS using a declarative language. Figure 1(b) depicts a screenshot of the *learner view*. The *C1* panel displays the database schema of a specific application of interest. A learner submits his/her query in the *C2* panel. Upon clicking on the Submit button, the *C3* and *C4* panels display the NL description in different presentation modes. Clicking on the Compare button shows the differences between NL descriptions generated by a rule-based and a deep learning-based techniques. The panel *C5* shows some example queries for learners to explore. Panels *C6-C7* support the question-answering framework of NEURON [11], which is beyond the scope of this demonstration. The *C8* panel is used to receive feedbacks from learners. Clicking on Switch DB (top-right corner) enables us to change the underlying RDBMS (*e.g.,* from PostgreSQL to SQL Server). Clicking on POEM

[1]Watt and Vodanovich [16] describe boredom as a dislike of or repetition of routine.

takes us to the *SME view* (*i.e.,* interface for SMEs), which we shall discuss in the next subsection.

3.2 POOL Module

Creating a QEP-to-NL description dataset to train a deep learning-based framework for generating NL descriptions of arbitrary QEPs is challenging in practice as the NL labeling needs to be performed by trained SMEs in order to ensure accuracy of the annotations. Given that there can be numerous QEPs in practice, it is prohibitively expensive to deploy such experts for labeling QEPs.

POOL (Physical Operator Object Language) addresses this challenge by providing a declarative interface to enable SMEs to create and manipulate the NL descriptions of physical operators in an RDBMS. All QEPs are essentially constructed from this set of operators, which is orders of magnitude smaller than a training set containing QEPs, making it viable to obtain the NL descriptions (*i.e.,* labels) from SMEs. The NL descriptions of operators in a QEP are stitched together automatically by the *boredom-conscious NL description generator module* to generate the description of a QEP.

The data model underlying POOL is called POEM (Physical Operator ObjEct Model), which is a simple and flexible graph model where all entities are objects. Each object represents a physical operator of a relational query engine. Each object has a unique *object identifier* (*oid*) from the type *oid*. Objects are either atomic or complex. Atomic objects do not have any outgoing edges. Each object is associated with the following attributes: *source*, *name*, *alias*, *defn*, *desc*, *type*, *cond*, and *target*. The *source* refers to the specific relational engine that an operator belongs to. We can create different operator objects for different RDBMS by changing the *source*. The *name* refers to the name of a physical operator in the source and the *type* captures whether it is an unary or binary operator. *Alias* is an optional alternative name for an operator. The *defn* attribute stores the definition of an operator. The *desc* attribute stores an NL description of the operation performed by an operator. There can be multiple *desc* associated with an object. The *cond* attribute takes a Boolean value to indicate whether a specified condition (*e.g.,* join condition) should be appended to the NL description of an operator. Values of all attributes are taken from the atomic type string (possibly empty). There is a directed edge between an object pair (p_a, p_c) iff p_a is used to describe p_c. For example, (p_{hash}, p_{hashjoin}) of PostgreSQL has a directed link as p_{hash} is used to describe a hash join. In this case, the *target* attribute value of p_{hash} is 'HASH JOIN'.

POOL supports SQL-like statements to manipulate the operator objects. For example, the COMPOSE statement can be used to generate the NL description *template* of an operator. For instance, the template of the HASH JOIN operator of PostgreSQL can be generated using it. Specifically, it combines the descriptions associated with the HASH and HASH JOIN operators to generate the following template: *"hash <T> and perform a hash join on table <R> and <T> under condition (<C>) to get the intermediate table <TN>"* where *"hash <T>"* is the *desc* value associated with the HASH operator. Note that each tag (*e.g.,* <T>, <C>, etc.) in *desc* acts as a place holder and has a specific meaning as reported in [15]. Particularly, this makes LANTERN orthogonal to any specific application domain. A place holder is replaced by a specific relation name, attribute name, or a predicate associated with a query on a database schema to generate an application-specific NL description.

Form-based interaction. We provide an interactive form that encapsulates aforementioned features. It provides basic operations over POEM, such as search, edit, add, and delete. These operations are converted internally into corresponding POOL statements.

3.3 Boredom-conscious NL Description Generator Module

Given an SQL query Q from a learner, this module is responsible for generating the natural language description of the QEP of Q. To this end, it realizes the following components.

QEPParser. When a user submits an SQL query, this module retrieves and parses the corresponding QEP to construct a *physical operator tree* (*operator tree* for brevity). A node in an operator tree contains relevant information associated with it such as the physical operator (*e.g.,* HASH JOIN), the name(s) of the relation(s) being processed by it, the alias given to the intermediate results (*e.g.,* subqueries), the column(s) used for grouping or sorting, subplan ids, access methods and predicates, and the number of rows left after the operation. Next, it records the physical operators involved in the QEP (if an operator appears multiple times in a QEP it is only counted once).

RULE-LANTERN. It realizes a rule-based framework that utilizes the narration of various operators defined using POOL to generate an NL description of the QEP of an SQL query [15]. It first extends the operator tree to a *language annotated operator tree* (LOT) by annotating the nodes with corresponding NL descriptions from the POEM store and assigning a unique *identifier* to the output of each operator (*i.e.,* intermediate results) so that it can be appropriately referred to in the translation. Then, it traverses the LOT in a post-order manner to generate a sequence of steps containing the NL description by replacing the place holders in NL templates with corresponding values.

NEURAL-LANTERN. Recall that the NL statements generated by RULE-LANTERN can be repetitive and lack variability that may cause boredom among learners [15]. NEURAL-LANTERN is a deep learning-based framework designed to mitigate this challenge. To address the paucity of training data, it adopts Kipf *et al.* [10] to generate a set of SQL queries given a particular schema and a database instance. A collection of QEPs corresponding to these queries is then generated. Each QEP is decomposed into a set of *acts*, each of which corresponds to a set of operators in an operator tree (*i.e.,* subtree). For each act, RULE-LANTERN is invoked to generate the corresponding NL description. Then for each result, it infuses language variability in the generated description by exploiting a group of paraphrasing tools [1–3] and pretrained word embeddings (*e.g.,* [5, 13]) to acquire a set of synonymous sentences. As a result, the number of training samples in our datasets is enlarged by approximately 3X.

The translation model of NEURAL-LANTERN follows the *Seq2Seq* structure [14]. The encoder RNN encodes each word in an act into the corresponding hidden state using an LSTM layer. It uses an LSTM decoder with an attention mechanism to let the decoder focus on the relevant portion of the encoder while generating a token. It adopts both static (*Word2Vec* and *GloVe* [12]) and contextual word embeddings (*ELMo* [13] and *BERT* [5]) in the decoder.

Boredom-conscious NL description generator. This component integrates the aforementioned components to implement a boredom-conscious NL description generation framework for a given QEP P. Specifically, the goal is to track the *repetition rate* of physical operators in a user's query history over T days (by default $T = 1$) and select an appropriate NL description generation scheme (either RULE-LANTERN or NEURAL-LANTERN) based on it. It computes the ratio of the number of operators in P to the total number of operators contained in all QEPs in a user's query history. If the ratio is below a predefined *repetition threshold* (40% by default), it invokes RULE-LANTERN to generate the NL description of P. Otherwise, it invokes NEURAL-LANTERN to generate it, which infuses variability in the description. This avoids a learner viewing repetitive text after submitting several queries within T days, thereby mitigating boredom as reported in a user study in [15].

3.4 Presentation Module

Finally, this module is responsible for displaying the generated NL description of a QEP in multiple modes as follows: (a) *Document view*: The NL description is displayed as a text document (*e.g.,* Panel *C3* in Figure 1(b)). (b) *Annotated visual tree view:* This mode *integrates* the visual tree view of a QEP with the NL description output. Specifically, the visual operator tree is shown by default and the NL description corresponding to each step is added as an annotation to the relevant node in the tree (Panel *C4*). A user can view the NL description of an operation by simply clicking on the corresponding node in the tree. (c) *Comparative view:* In this mode, a learner can comparatively view the differences between the document views generated by RULE-LANTERN and NEURAL-LANTERN. The differences in the descriptions are highlighted in the text.

4 RELATED SYSTEMS

The problem of translating natural language queries to SQL has been studied for decades [9]. LANTERN compliments these efforts by providing a natural language description of a QEP. Most germane to this work is NEURON [11], which is a rule-based system to generate NL descriptions of QEPs. Broadly, NEURON only realizes the second design principle in Section 2 whereas LANTERN achieves all four. More specifically, LANTERN differs from it in the following key ways. Firstly, instead of exploiting a purely rule-based solution that may give rise to boredom among learners due to similar/same descriptions, LANTERN integrates a rule-based and a deep learning-based solutions to inject variability in the NL descriptions opportunely. Secondly, NEURON is tightly integrated with PostgreSQL and hence is not portable across RDBMS. LANTERN implements a declarative framework called POOL for labeling physical operators of different RDBMS to enhance its portability. Thirdly, [11] presented the NL descriptions of queries in the *document view* mode only. LANTERN is more flexible as it enables a multi-faceted view of the NL descriptions. Consequently, all key modules of LANTERN are novel and have not been demonstrated in any prior venues.

5 DEMONSTRATION PLAN

LANTERN is implemented in Python. Our demonstration will make use of the TPC-H benchmark and IMDB datasets as representatives of two different applications. We shall use two different RDBMS, PostgreSQL and SQL Server, to demonstrate the portability of LANTERN. By default, LANTERN uses PostgreSQL. The datasets are replicated on both these RDBMS for ease of demonstration. The audience can pose their own ad-hoc queries on these datasets. The goal of our demonstration is to allow the audience to experience the following interactive features of LANTERN. A short **video** of LANTERN is available at https://youtu.be/jv7kJe5Gxo0.

Boredom-conscious NL description generation and exploration. An audience can generate and explore an NL description of the QEP of his/her query through the *learner view* (Figure 1(b)). One can select a database schema in the *C1* Panel and input an SQL query in *C2*. She may also select one from the example queries in *C5*. The *C3* Panel displays the NL description of the corresponding QEP in document view mode. In particular, the audience will be encouraged to fire several queries and peruse the NL descriptions to appreciate the generation of similar text by RULE-LANTERN as well as the injection of variability in these descriptions by NEURAL-LANTERN. One may click on the Compare button to visualize the differences between RULE-LANTERN and NEURAL-LANTERN outputs. One may also view the NL descriptions in annotated visual tree mode (*C4*). One can click on the Switch DB link to switch to a different RDBMS (*e.g.,* from PostgreSQL to SQL Server). We believe all these interactions will trigger interesting discussions on the benefits of multi-faceted presentation of NL descriptions to aid learning as well as the impact of similar descriptions on boredom.

Label generation and portability using POOL interface. We demonstrate how one can declaratively create labels for physical operators in a different RDBMS (SQL Server) using the form-based interface of POOL. Specifically, clicking on the POEM link takes a user to the POEM GUI. The audience can create and manipulate NL descriptions of different physical operator objects associated with the specific RDBMS either using the form-based interface or by directly writing statements using POOL. After this, we shall return back to the *learner view*. We shall input queries on the database applications hosted on SQL Server now and generate the corresponding NL descriptions of QEPs. This will demonstrate the portability of LANTERN across different RDBMS.

Acknowledgements. Peng Chen and Hui Li are supported by National Natural Science Foundation of China (No. 61972309), XD-Inspur DB Innovation Lab grant, Key Scientific Research Program of Shaanxi Provincial Department of Education (No. 20JY014). Sourav S Bhowmick, Shafiq Joty, and Weiguo Wang are supported by AcRF Tier-1 Grant 2018-T1-001-134.

REFERENCES

[1] Paraphrasing tool. https://paraphrasing-tool.com/.
[2] Prepostseo paraphrasing tool. https://www.prepostseo.com/paraphrasing-tool.
[3] Quillbot paraphraser. https://quillbot.com/.
[4] J. T. Cacioppo and R. E. Petty. Effects of Message Repetition and Position on Cognitive Response, Recall, and Persuasion. *Journal of Personality and Social Psychology* ,37, 1: 97-109, 1979.
[5] J. Devlin, M. Chang, K. Lee, K. Toutanova. Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv e-prints*, 2018.
[6] M. R. Hastall and S. Knobloch-Westerwick. Severity, Efficacy, and Evidence Type as Determinants of Health Message Exposure. *Health Communication*, 28, 4: 378-388, 2013.
[7] G. Hervet, K. Guerard, S. Tremblay, M. Saber Chtourou. Is Banner Blindness Genuine? Eye Tracking Internet Text Advertising. *Applied Cognitive Psychology*, 25, 5: 708-716, 2011.
[8] Z. Ives, et al. VLDB Panel Summary: "The Future of Data(base) Education: Is the Cow Book Dead?". *SIGMOD Rec.*, 50(3), 2021.
[9] H. Kim et al. Natural Language to SQL: Where Are We Today? *In PVLDB*, 13(10), 2020.
[10] A. Kipf, et al. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. *In CIDR*, 2019.
[11] S. Liu, et al. NEURON: Query Optimization Meets Natural Language Processing For Augmenting Database Education. *In SIGMOD*, 2019.
[12] J. Pennington, R. Socher, C. Manning. Glove: Global Vectors for Word Representation. *In EMNLP*, 2014.
[13] M. Peters, et al. Deep Contextualized Word Representations. *In NAACL*, 2018.
[14] I. Sutskever, et al. Sequence to Sequence Learning with Neural Networks. *In NeurIPS*, 2014.
[15] W. Wang, S. S. Bhowmick, et al. Towards Enhancing Database Education: Natural Language Generation Meets Query Execution Plans. *In SIGMOD*, 2021.
[16] J. D. Watt , S. J. Vodanovich. Boredom Proneness and Psychosocial Development. *The Journal of Psychology*, 133/3, pp. 303-314, 1999.