PreQR: Pre-training Representation for SQL Understanding

Xiu Tang, Sai Wu*, Mingli Song, Shanshan Ying, Feifei Li, Gang Chen

Zhejiang University & Alibaba Group

AZFT (Alibaba-Zhejiang University)
Learning-based Database Optimization

(a) Data Model

- Query
  - SQLs
  - Query Parser
    - predicates
  - Data Sampling
    - values/tuples
    - encoding
  - Query Model
    - Cardinality
    - Cost

(b) Query Model

- Query
  - SQLs
  - Query Feature Extractor
    - predicates
  - Query Encoder
    - query encoding
  - Query Model
    - Cardinality
    - Cost
Learning-based Database Optimization

(a) Data Model

Query
SQLs
Query Parser
predicates
Data Sampling
values/tuples
encoding
Query Model
Cardinality
Cost

(b) Query Model

Query
SQLs
Query Feature Extractor
predicates
Query Encoder
query encoding
Query Model
Cardinality
Cost
Previous Approach: One-hot Encoding

- **SQL structure information:**
  Encoding simply concatenates the encoding of all clauses in the query.

- **Database schema information:**
  All tables and columns use an independent one-hot encoding.

- **Database column value distribution information:**
  All values in SQL are normalized to $[0,1]$.
Previous Approach: One-hot Encoding

- **SQL structure information:**
  Encoding simply concatenates the encoding of all clauses in the query.

- **Database schema information:**
  All tables and columns use an independent one-hot encoding.

- **Database column value distribution information:**
  All values in SQL are normalized to $[0, 1]$. 

---

**Input SQL:**
```
SELECT t.id
FROM title t, movie_companies mc
WHERE t.id = mc.movie_id
AND t.product_year > 2010
AND mc.company_id = 5
```

**One-hot encoding:**
- Column set: $\{[0 0 0 0 0 1], [1 0 0 1 0 0 0 0]\}$
- Table set: $\{[0 1], [1 0]\}$
- Predicate set: $\{[0 0 0 0 1 0 1 0 1 0 0 0 0], [1 0 0 0 0 0 0 1 0.14]\}$

**Drawbacks:**
Ignoring query structure, database schema, distribution variance.
• SQL structure information:
  Encoding simply concatenates the encoding of all clauses in the query.

• Database schema information:
  All tables and columns use an independent one-hot encoding.

• Database column value distribution information:
  All values in SQL are normalized to [0,1].
Previous Approach: One-hot Encoding

- **SQL structure information:**
  Encoding simply concatenates the encoding of all clauses in the query.

- **Database schema information:**
  All tables and columns use an independent one-hot encoding.

- **Database column value distribution information:**
  All values in SQL are normalized to \([0, 1]\).
Previous Approach: Pretrained Language Model

- The language representation has been well studied by work on the NLP.
- However, SQL incurs **new challenges:**
  - **Semantically equivalent:**
    - query $q_3$ and $q_1$, which can be easily identified by their query structures;
    - query $q_5$ and $q_4$, which can be discovered via involved schema information.

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>SELECT name FROM user WHERE rank IN ('adm','sup')</td>
</tr>
<tr>
<td>$q_2$</td>
<td>SELECT SUM(balance) FROM accounts</td>
</tr>
<tr>
<td>$q_3$</td>
<td>SELECT name FROM user WHERE rank = 'adm' UNION SELECT name FROM user WHERE rank = 'sup'</td>
</tr>
<tr>
<td>$q_4$</td>
<td>SELECT SUM(balance) FROM accounts WHERE user_id IN (SELECT user_id FROM user WHERE rank = 'adm')</td>
</tr>
<tr>
<td>$q_5$</td>
<td>SELECT SUM(accounts.balance) FROM accounts, user WHERE accounts.user_id = user.id AND user.rank = 'adm'</td>
</tr>
</tbody>
</table>

$\rightarrow$ Logically Same  $\rightarrow$ Query Dependent
Previous Approach: Pretrained Language Model

- The language representation has been well studied by work on the NLP.
- However, SQL incurs **new challenges:**
  - **Semantically equivalent:**
    - query $q_3$ and $q_1$, which can be easily identified by their query structures;
    - query $q_5$ and $q_4$, which can be discovered via involved schema information.
Previous Approach: Pretrained Language Model

- The language representation has been well studied by work on the NLP.
- However, SQL incurs **new challenges**:
  - **Semantically equivalent**:
    - query $q_3$ and $q_1$, which can be easily identified by their query structures;
    - query $q_5$ and $q_4$, which can be discovered via involved schema information.
Previous Approach: Pretrained Language Model

• The language representation has been well studied by work on the NLP.

• However, SQL incurs **new challenges**:
  • **Semantically equivalent:**
    • query $q_3$ and $q_1$, which can be easily identified by their query structures;
    • query $q_5$ and $q_4$, which can be discovered via involved schema information.
Introducing PreQR

• **PreQR**: Pretraining Query Representation.

• By pretraining query representation, **PreQR**:
  - integrates the database schema, query structure and content knowledge.
  - only needs to be trained once for a database and can be used in various learning tasks.
  - performances on various database tasks obtain a significant improvement.
Introducing PreQR

- **PreQR**: Pretraining Query Representation.
- By pretraining query representation, **PreQR**:
  - integrates the database schema, query structure and content knowledge.
  - only needs to be trained once for a database and can be used in various learning tasks.
  - performances on various database tasks obtain a significant improvement.
• The **input embedding** represents the query structure via matching automaton states.

• The **query-aware schema** use a graph-structured model to encode SQL-related schema information.

• The **SQL BERT encoder** leverages the attention mechanism to identify the query-aware structural and schema information in an ad-hoc way.
• PreQR transforms the query structure into a finite-state automaton (FA), which is a machine with a finite number of states.

• Automata can recognize syntactically well-formed strings to represent the semantic structure of SQL.

<table>
<thead>
<tr>
<th>Input</th>
<th>Queries $q_1$ in Section 1</th>
<th>Queries $q_3$ in Section 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automaton Matching</td>
<td>START $a_0$ → SELECT $a_1$ → Column $a_2$ → FROM $a_3$ → Table $a_4$ → WHERE $a_5$ → Column $a_6$ → $=$ $a_7$ → String $a_8$ → END $a_9$</td>
<td>UNION $a_{10}$</td>
</tr>
<tr>
<td>SQL State Embedding</td>
<td>$a = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_9, a_3, a_{11})$</td>
<td>$a = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_8, a_9, a_{10}, a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_8, a_9, a_{11})$</td>
</tr>
</tbody>
</table>
PreQR Input Representation
**Tables:** \( T = \{ \text{Title}, \text{Movie keyword}, \text{Cast info}, \text{Movie info}, \text{Movie companies}, \ldots \} \)

**Columns:** \( C_{\text{Title}} = \{id, \text{title}, \text{kind id}, \text{production year}, \ldots\} \)

\( C_{\text{movie companies}} = \{\text{movie id}, \text{company id}, \text{company type id}, \ldots\} \)

**Foreign:** \( F = \{(\text{title.id}, \text{movie companies.movie id}), (\text{title.id}, \text{movie info.movie id}), \ldots\} \)
Query:  \[ q = \text{"SELECT COUNT(*) FROM title } t, \text{ movie_companies mc WHERE } t.id = mc.movie_id \text{ AND } t.production_year > 2010 \text{ AND mc.company_id = 5"} \]
**Trm_g Module in PreQR**

- *Trm_g* architecture is a variant of the Transformer from BERT.

- The *Trm_g* model includes the original Transformer *Trm* (black rectangle) and our query-aware sub-graph Transformer *Trm’* (red rectangle).

- PreQR augments each word with the graph structure of the schema items that it is linked to.
**Trm\_g Module in PreQR**

- *Trm\_g* architecture is a variant of the Transformer from BERT.
- The *Trm\_g* model includes the original Transformer *Trm* (black rectangle) and our query-aware sub-graph Transformer *Trm*' (red rectangle).
- PreQR augments each word with the graph structure of the schema items that it is linked to.
Trm\_g Module in PreQR

- *Trm\_g* architecture is a variant of the Transformer from BERT.

- The *Trm\_g* model includes the original Transformer *Trm* (black rectangle) and our query-aware sub-graph Transformer *Trm’* (red rectangle).

- PreQR augments each word with the graph structure of the schema items that it is linked to.
Extensibility

- Case 1: The distribution of data changes significantly.
- Case 2: If the database schema is updated, we need to update the schema graph model $G_s$.
- Case 3: When query patterns change, we may need to update the FA to handle new queries.
- Case 4: Training a new embedding model for a database from scratch.

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>Incremental learning for the last layer of SQLBERT</td>
<td>15min</td>
</tr>
<tr>
<td>Case 2</td>
<td>Incremental Learning for the Schema2Graph part</td>
<td>3.5h</td>
</tr>
<tr>
<td>Case 3</td>
<td>Incremental learning for the Input Embedding module</td>
<td>6.7h</td>
</tr>
<tr>
<td>Case 4</td>
<td>Train from scratch</td>
<td>18.3h</td>
</tr>
</tbody>
</table>
Experiment Highlight

PreQR handles various downstream tasks:

- **Query Clustering:**
  Comparing with five approaches to measure pairwise similarity between queries.

- **SQL-to-Text Generation:**
  Comparing the encoding of PreQR model against the Seq2Seq, Tree2Seq and Graph2Seq.

<table>
<thead>
<tr>
<th>SQL</th>
<th>SELECT opponent WHERE points &lt; 18 AND November &gt; 11;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq2Seq</td>
<td>What is the opponent when the points are less than 18 with the November is more than 11?</td>
</tr>
<tr>
<td>PreQR</td>
<td>Which opponent has the points less than 18, and the November more than 11?</td>
</tr>
</tbody>
</table>
Experiment Highlight

• Query Cardinality and Cost Estimation:
  Comparing with a conventional method (PostgreSQL), the query-based learning models (MSCN and LSTM), and a data-based learning model (NeuroCard).

• The experimental results showed that by replacing the encoders of existing models with PreQR encoding, performances on various database tasks obtain a significant improvement.
PreQR

- PreQR: towards pre-training SQL embedding.

Xiu Tang
- Email: tangxiu@zju.edu.cn