arXiv:2303.14777v1 [cs.DB] 26 Mar 2023

基于生成对抗网络的查询生成

Weihua Sun、Run-An Wang 和 Zhaonian Zou

中国哈尔滨工业大学

摘要。 数据库系统中的许多问题，例如基数估计、数据库测试和优化器调优，都需要大量查询负载作为数据。然而，由于用户隐私限制或数据库访问频率较低，通常很难从用户处获取大量真实查询。查询生成是解决这一问题的方法之一。现有的查询生成方法，例如随机生成和基于模板的生成，没有考虑生成的查询与现有查询之间的关系，甚至会生成语义不正确的查询。本文提出一种基于生成对抗网络（GAN）的查询生成框架，用于生成与给定查询负载相似的查询负载。在我们的框架中，我们使用语法解析器将查询转换为解析树，并遍历该树，以获取与查询对应的产生式规则序列。GAN 的生成器以固定的先验分布为输入，并输出查询序列；判别器则以真实查询和生成器生成的假查询为输入，并输出梯度以指导生成器学习。此外，我们在生成过程中加入上下文无关文法和语义规则，从而确保生成的查询在句法和语义上均正确。我们在真实世界数据集上进行实验评估，结果表明，该方法能够生成分布与给定查询负载相似的新查询负载，且生成的查询在句法上正确，不存在语义错误。生成的查询负载被用于下游任务，结果显示，使用我们的方法扩展查询负载后训练的模型性能显著提升。

关键词： 查询生成 · 生成对抗网络 · 上下文无关文法 · 解析树 · 语义规则

1 引言

在数据库系统中，查询负载是指具有特定分布的一组查询。数据库系统中的许多问题，例如基数估计 [5],数据库测试 [21], 视图/索引选择 [19], 数据库分区 [4], 和优化器调优 [11], ，都需要大量查询负载参与方法的构建或测试。例如，作为查询优化中的一个基础问题，基数估计用于预测查询实际执行前其结果中将出现多少个元组，

即基数。基于学习的查询驱动的基数估计器使用机器学习模型将查询映射到其基数。此类估计器需要使用大量高质量查询及其基数来训练模型。如果训练集中没有足够的查询，模型的泛化性能往往较差。此外，当新的查询负载到来时，估计器需要进行更新，以确保模型准确且保持最新。

查询负载在解决数据库系统中的各种问题时发挥着非常重要的作用。然而，除了大型 IT 企业，实际上很难从用户那里获得大量真实查询，因为这涉及用户隐私，或者数据库访问频率较低。查询生成，即为数据库生成查询的技术，是解决这一问题的一种方法。查询生成既可以不依赖任何给定的查询负载，也可以依赖已知的查询负载。现有的查询生成方法，例如随机生成 [15,1] 和基于模板的生成 [3,15], ，并未考虑生成的查询负载与给定查询负载之间的关系，而且生成的查询甚至可能存在句法或语义错误。因此，确实有必要设计一种依赖查询负载的方法，生成与给定查询负载的分布非常相似的查询负载，从而扩展给定的查询负载。

查询生成与人工智能领域中的图像或文本生成有一些相似之处。随着机器学习的发展，越来越多的研究开始探索使用深度学习的生成模型。生成对抗网络（GAN） [6] 和变分自编码器（VAE） [9], 等生成模型已在图像生成 [8] 和文本生成 [18]中取得了良好效果。在一些带有语法的文本生成任务中，TreeGAN [12] 和其他研究 [17] 进行了初步尝试。然而，这些方法尚未针对 SQL 等特定语言进行研究，也未将领域知识融入生成过程。

本文提出了一种基于 GAN 的查询生成框架，用于生成与给定查询负载相似的查询负载。在该框架中，我们使用语法解析器将查询转换为解析树，并遍历该树，以获取与查询对应的产生式规则序列。GAN 的生成器以固定的先验分布为输入，并输出查询序列；判别器则以真实查询和生成器生成的假查询为输入，并输出梯度来指导生成器学习。我们还定义了一个简化版的 SQL 上下文无关文法（上下文无关文法），用于指导查询生成。基于 SQL 上下文无关文法构建的掩码矩阵可确保生成的查询在句法上正确。此外，我们总结了常见的语义错误，并实现了基于语义规则的生成约束，以避免生成语义错误的查询。在生成过程中，我们跟踪当前序列的语义状态，并屏蔽会导致语义错误的操作。为了使模型更好地学习查询分布，我们将查询基数、估计成本等查询特征信息加入查询序列。相应地，我们修改了 GAN 的判别器，使其能够捕获这些信息。

本文的贡献可总结如下：
1. 我们提出了一种基于 GAN 的查询生成方法，该方法可以学习给定查询负载的分布并生成新的查询负载。2. 我们定义了一种简单的 SQL上下文无关文法，并将其应用于查询生成过程，以确保生成的查询句法正确。我们总结了常见的语义错误，并实现了基于语义规则的生成约束，以确保生成的查询在语义上正确。3. 我们对查询的特征信息进行编码，并将其加入生成过程，以提高生成查询的质量。4. 实验表明，我们的模型能够生成高质量的查询，且生成的查询负载与给定的查询负载分布相似。我们将生成的查询负载应用于基数估计任务，结果表明，使用生成扩充的查询负载作为训练数据，能够显著提升模型的效果。

2 问题陈述

设 $D = \{R_1, R_2, \ldots, R_t\}$ 是一个关系数据库，其中 $R_1, R_2, \ldots, R_t$ 是关系。给定在 D 和 $m \in N$ 上的输入查询负载 $Q = \{q_1, q_2, \ldots, q_n\}$，与查询负载相关的查询生成问题旨在生成另一个查询负载 $Q' = \{q'_1, q'_2, \ldots, q'_m\}$，使得 Q' 与 Q 非常相似。对 Q 和 Q' 之间相似度的定义对查询生成至关重要，具体如下：

查询 q 是一条 SQL语句。可以通过许多特征来刻画查询 q。简单特征包括 q 中涉及的表、连接、谓词和属性的数量；复杂特征包括 q 的基数、由数据库管理系统的查询优化器估算的 q 成本，等等。令 f 表示查询 q的特征 $f(q)$的值。对于查询负载 X，所有查询 $q \in X$ 的 $f(q)$的分布，可以从特征 f的角度刻画 X 的某个特定方面。令 p_X^f 表示该分布的概率密度函数（pdf）。因此，我们可以通过 p_Q^f与 $p_{Q'}^f$之间的散度，按照特征 f 评估输入查询负载 Q 与生成的查询负载 Q' 之间的距离，即：

$$d^f(Q, Q') = \int_x \left| p_Q^f(x) - p_{Q'}^f(x) \right| dx, \tag{1}$$

其中， x 取自所有 $q \in X$的 $f(q)$所有可能值的集合。

设 $F = \{f_1, f_2, \ldots, f_k\}$ 为用于描述查询特征的集合。所有查询 $q \in X$ 的 $f_1(q)$ $f_2(q) \ldots, f_k(q)$ 的联合分布是对 X 的刻画。我们用 p_X^F 表示该联合分布的概率密度函数。通常，输入查询负载 Q 与生成的查询负载 Q'之间的距离可通过以下方式评估：

$$d^F(Q, Q') = \int_{x_k} \int_{x_{k-1}} \cdots \int_{x_1} \left| p_Q^F(x_1, x_2, \ldots, x_k) - p_{Q'}^F(x_1, x_2, \ldots, x_k) \right| dx_1 d_{x_2} \ldots dx_k, \tag{2}$$

其中，x_i 取自所有 $q \in X$对应的 $f_i(q)$ 的所有可能值的集合。

因此，本文要解决的问题可以正式表述如下：给定 D 上的查询负载 $Q = \{q_1, q_2, \ldots, q_n\}$、$m \in N$ 以及一组查询特征 F，我们希望生成一个查询负载 $Q' = \{q'_1, q'_2, \ldots, q'_m\}$，使 Q 与 Q' 之间的距离 $s^F(Q, Q')$ 最小。此外，生成的查询 q'_1，q'_2、……、q'_m 必须在句法和语义上均正确。

3 我们的方法

本文提出了一种基于生成对抗网络的查询生成方法。模型的架构如图 1 所示。首先，我们对给定的 SQL 查询 Q 进行预处理（第 3.1 节），将每条 SQL 查询转换为解析树，并遍历解析树以获得对应的产生式规则序列（第 3.2 节）。接下来，我们将产生式序列作为训练数据输入模型，经过充分训练的生成器便能够捕获查询的分布信息，并生成与给定查询负载 Q 相似的查询（第 3.3 节）。在训练和推理阶段，我们使用句法和语义规则约束生成器，使其生成正确的查询（第 3.4 节）。最后，模型将生成的产生式序列转换为 SQL 查询。

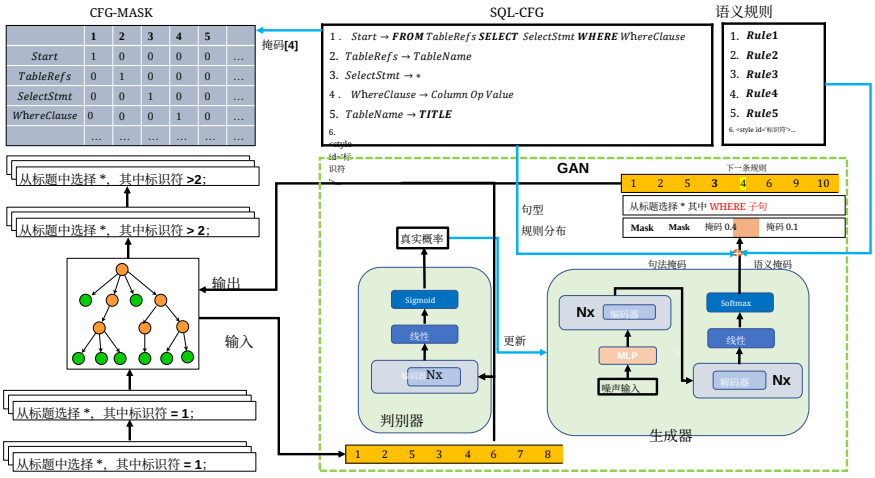


图 1。模型架构。

3.1 预处理

在我们的方法中，我们首先对给定查询负载 Q 中的查询进行预处理。预处理包括以下步骤：

重构查询 我们调整查询结构，使句法和语义规则更易于设计。生成器支持的查询结构为选择字段 来源字段 WHERE 子句 分组子句 HAVING 子句，我们将其调整为来源字段 选择字段 WHERE 子句 HAVING 子句 分组子句。我们进行了两项调整：首先，将来源字段移到查询开头，因为来源字段限定了查询中涉及的关系。其次，将分组子句移到末尾，因为选择字段和 HAVING 子句中出现但未进行聚合的属性需要在分组子句中分组。

缩小动作空间 查询中通常包含许多常量值。这些常量会使查询生成的动作空间变得极其庞大。为缩小动作空间，我们为 Q 中查询涉及的每个属性构建直方图，即将属性的定义域划分为多个桶，并用不同的哈希键表示每个桶。然后，将查询中的每个常量替换为其所属桶的哈希键。

按照上述方法对查询进行预处理后，查询的语义不会改变。在我们方法的下一步中，也以这种形式生成查询。要将预处理后的查询还原为可执行的 SQL，只需两步：第一步还原来源字段和分组子句；第二步根据哈希键从哈希桶中随机抽取一个值，并用该值替换哈希键。

3.2 语法引导

常见的文本生成工作会直接生成序列，而不考虑序列的结构和语法。如果生成器能够按照语法生成序列，我们就能得到语法正确的文本序列。本节将介绍如何将查询转换为树结构，以及如何基于语法生成查询。

SQL 的语法可以用上下文无关文法（CFG）表示。我们使用 SQL 的 CFG 来指导查询生成。CFG 定义如下：

$$CFG = \{V, T, P, S\}$$

其中， V 是非终结符集合， T 是终结符集合， P 是形式为 $P \to \gamma$ 的产生式集合，其中 $P \in V$， $\gamma \in (V \cup T)^*$， S 是语法的开始符号。在我们的工作中，我们为 SQL 构建了一个基本的 CFG。该语法的简化版本见表 1。

算法 1 展示了如何将查询负载转换为产生式序列。对于每个查询，模型使用解析器将其转换为解析树，然后对该树进行深度优先、从左到右的遍历。在遍历过程中：如果遇到非终结符，则将其用作产生式的头部，并将其子节点用作产生式的体部，然后将组合后的产生式添加到产生式序列中。最后，输出与输入查询负载对应的产生式序列。图 2 展示了解析

表 1. 简化的 SQL 上下文无关文法

0:
$Start \rightarrow \boldsymbol{FROM}\ TableRefs\ \boldsymbol{SELECT}\ SelectStmt\ \boldsymbol{WHERE}\ WhereClause$
1: $TableRefs \rightarrow TableName$ 2: $SelectStmt \rightarrow *$ 3:
$WhereClause \rightarrow Column\ Op\ Value$ 4: $TableName \rightarrow \boldsymbol{TITLE}$ 5:
$Column \rightarrow \boldsymbol{ID}$ 6: $Op \rightarrow =$ 7: $Value \rightarrow$ 1 8: $\cdots$

查询“FROM TITLE SELECT * WHERE ID $=$ 1”的解析树，通过遍历解析树，我们可以得到生成的序列 [0,1,4,2,3,5,6,7]。

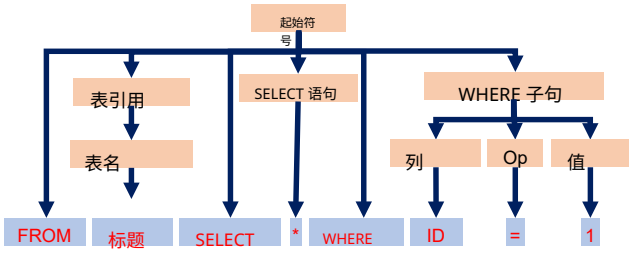


图 2. “FROM TITLE SELECT * WHERE ID $=$ 1”的解析树

给定一个上下文无关文法 $\{V, T, P, S\}$，推导过程是将形式为 $\alpha P \beta$ 的句型转换为 $\alpha \gamma \beta$，当且仅当存在产生式 $P \rightarrow \gamma$。直观而言，上下文无关文法从开始符号出发，逐步将产生式应用于推导过程：找到头部与当前句型中的最左非终结符相匹配的产生式，并用该产生式的体替换匹配的非终结符，直到句型中不再有非终结符。最终，我们得到一个符合该上下文无关文法的字符串。

算法 2 实现了从产生式序列到查询的转换过程。对于每个产生式序列，模型都会维护一个符号栈和一个文本序列。符号栈用于存储待推导的句型，初始时栈中仅包含语法的开始符号。栈中的符号逐个出栈；如果出栈的符号是非终结符，则应用序列中的产生式，完成一次推导过程。如果遇到终结符，则将其添加到文本序列中。当符号栈中没有符号时，文本序列就是我们需要的查询。

算法 1: 从查询到产生式序列

输入: 查询负载 $Q = \{q_1, q_2, \ldots, q_n\}$，SQL-CFG $G = \{V, T, P, S\}$
输出: 产生式序列 $S = \{s_1, s_2, \ldots, s_n\}$

```
函数Main:
    for q_i 于 Q执行
        树 = 解析器(G, q_i)// 使用 LALR解析器
        s_i =深度优先遍历(树)
        S = S ∪ {s_i}
    返回 S
函数深度优先遍历(节点):
    s = {}
    CheckNode(节点, s)
    对于child in node.children 执行
        CheckNode(child.root,s)
    返回 s
函数CheckNode(node,s):
    如果 node.type 是非终结符 则
        头部 =node
        主体 =node.children.data
        s = s ∪ {head → body}
    返回 s
```

我们的查询生成方法的一个重要思想是利用产生式序列来推导查询。对于每个查询 $q_i \in Q$，我们使用算法 1 得到产生式序列 S_i 。产生式序列 S_i 部分揭示了 q_i 的语法结构。我们的查询生成模型旨在生成查询负载 Q 对应的查询，其产生式序列与该负载中查询的产生式序列 S_1， S_2，... ， S_n 相似。

3.3 生成模型

我们的查询生成框架以生成模型为核心，该模型由生成对抗网络（GAN）表示。GAN 是一种无监督机器学习框架，由生成器网络（G）和判别器网络（D）组成。生成器 G 将固定先验 z 作为输入，并输出一个可转换为 SQL 查询的产生式序列。判别器 D 是一个二分类网络，它以产生式序列作为输入，并输出该输入序列来自输入查询负载而非 G 的概率。我们希望 G 生成的序列尽可能与给定查询负载中的序列相似，因此需要性能良好的 D 来指导 G 的工作。因此，GAN 的优化目标为:

$$\min_G \max_D V(D,G) = \underset{x \sim p_{data}(x)}{E} [\log D(x)] + \underset{z \sim p(z)}{E} [\log (1 - D(G(z)))], \quad (3)$$

算法 2: 从产生式序列到查询

输入: 产生式序列 $S = \{s_1, s_2, \ldots, s_n\}$
输出: 查询负载 $Q = \{q_1, q_2, \ldots, q_n\}$

```
Q = {}
对于 s_i 在 S 中
    q_i = {}
    符号栈 = [起始符号]// 从上下文无关文法的开始符号推导
    产生式栈 = reversed(s_i)// 从第一个产生式开始
    当符号不为空时
        符号 =符号栈.pop()
        如果符号是终结符，则
            q_i = q_i ∪ {symbol}
            继续
        else
            产生式 = 产生式栈.pop()
            对于reversed(production.body)中的symbol 执行
                symbols.push(symbol)
    Q = Q ∪ {q_i}
返回 Q
```

其中， x 是从 Q中查询的产生式序列中选取的一个产生式序列。对于最优 D，式（3）可以最小化生成序列与给定序列之间的 JS散度。为了提高生成序列的质量，我们采用 SeqGAN [18]中提出的方法训练 GAN。在本节剩余部分，我们将讨论生成器和判别器的具体细节。

生成器 受自然语言处理中注意力机制成功的启发 [16], ，我们使用 Transformer 作为生成器来生成可转换为查询的产生式序列。Transformer 的编码器将一个固定的先验$z \sim p(z)$作为输入，对其进行编码并传递给解码器。在每一步 t，解码器的输入为 $\mathbf{x}_t = [\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_{t-1}]$，其中 $\mathbf{p}_i$和 $i < t$是步骤 i 中所选产生式的嵌入向量。解码器的输出为 o_t，网络使用 Softmax 输出层将 o_t 映射为概率向量 $V^t_{1:\ |P|}$

$$V^t_{1:|P|} = softmax(W_v o_t + b_v). \tag{4}$$

在每个时间步 t，模型根据向量 $V^t_{1:\ |P|}$从产生式集合 P中采样一个产生式 $\mathbf{p}_t$。在此过程中，我们应用所选产生式进行推导，直到所有非终结符都被终结符替换，从而得到可转换为查询的产生式序列。

判别器 与生成器一样，我们使用 Transformer 编码器作为判别器。编码器将产生式序列作为输入，并生成

一个输出序列。我们对输出序列执行平均池化操作，以获得输入序列的嵌入表示 o_T。然后对其应用 Sigmod 激活函数，得到该序列来自输入查询负载的概率 $\hat{y}$：

$$\hat{y} = \sigma(W_o \cdot o_T + b_o).$$

为了使模型更好地学习查询的分布，也会将查询基数和成本等特征信息输入判别器。实践中，我们根据产生式序列重构 SQL，然后将 SQL语句输入数据库执行，以获取特征信息。特征信息经过简单归一化和编码后，添加到产生式序列的开头。因此，判别器的输入变为$\mathbf{x} = [\mathbf{f}_1, \cdots, \mathbf{f}_n, \mathbf{p}_1, \cdots, \mathbf{p}_T]$，其中$\mathbf{f}_i$是归一化后的特征信息。

3.4 句法和语义约束

请注意，在推导过程中，我们总是选择当前句型中的最左非终结符 v ，并查找其头部也为v的产生式。在上一节中，我们根据式 4 中的 $V^t_{1:|P|}$ 在整个产生式空间中采样产生式。然而，模型可能会选择头部不是 v 的产生式，从而导致推导过程出错。例如，当前句型为“FROM TITLE SELECT ***WHERE 子句**”，最左非终结符是**WHERE 子句**，但生成器选择了产生式$TableName \rightarrow Title$，导致推导失败。

为避免这种情况，我们构建了一个基于上下文无关文法的掩码矩阵 $M \in \{0,1\}^{|V|\times|P|}$，其中$|V|$ 和 $|P|$ 分别是非终结符和产生式的数量。具体而言，对于非终结符 $v \in V$，其掩码向量表示为 $M(v)$，其中以 v 为头部的产生式标记为 1，其余产生式标记为 0。例如，上下文无关文法开始符号 $M(\boldsymbol{Start})$ 的掩码向量为 (1，0，0，...，0)，因为表 1 中只有第 0 个产生式以 $\boldsymbol{Start}$ 为头部。在步骤 t，修改后的概率向量变为

$$\dot{V}^t_{1:|P|} = V^t_{1:|P|} \odot M(v), \tag{5}$$

其中 v 是当前句子中最左侧的非终结符，$\odot$ 表示逐元素相乘，而 $M(v)$则用于屏蔽无效产生式。

尽管基于上下文无关文法的掩码矩阵可以确保查询的句法正确性，但仍可能出现语义错误。例如，我们可能会生成类似于 "SELECT Name FROM Title WHERE Name = A" 的查询。虽然该查询看起来没有语法错误，但如果 Title 关系中不存在 Name 属性，该查询就无法正确执行。在我们的工作中，我们实现了基于语义规则的生成约束，以确保语义正确性。

在生成过程中，我们会跟踪当前序列的语义状态，并屏蔽会导致语义错误的产生式。在步骤 0，我们初始化一个语义掩码 $m_s = \{1\}_{1:|P|}$。每当生成器选择一个产生式时，

我们会根据语义规则和当前序列更新 m_s。当生成器即将选择一个产生式时，其概率向量会修改为:

$$\ddot{V}^t_{1:|P|} = \dot{V}^t_{1:|P|} \odot m_s \tag{6}$$

其中，$\dot{V}^t_{1:\ |P|}$ 是公式（5）给出的概率向量。在该向量中，会导致语法错误和语义错误的产生式被标记为 0，生成器会忽略这些产生式。

示例 *1.* 当我们生成部分查询"FROM Title"时，会更新语义掩码，将 Title 关系中的属性标记为 1，并将其他属性标记为 0。在生成剩余查询的过程中选择关系中的属性时，我们只选择 Title 关系中存在的属性。这样可以确保查询语义正确。

4 评估

在本节中，我们进行实验，以评估生成的查询负载与给定查询负载之间的相似度。我们还将生成的查询用于基数估计任务，以评估这些查询的可用性。

4.1 实验设置

数据集 我们在真实世界数据集 IMDB 上进行了评估。我们创建了 IMDB 数据库，并使用四个随机初始化的生成器合成了 4 个查询负载。每个负载包含 2000 条查询，复杂度各不相同。这 4 个查询负载的信息见表 2。

查询负载	嵌套	聚合	字符串	查询长度		数据库模式	Size
SPJ-N	No	No	No	[0,40]		6 张表和 9 列 2000	
AN-N	Yes	Yes	No	[0,40]		6 张表和 9 列 2000	
AN-S	Yes	Yes	Yes	[0,40]		21 张表和 22 列 2000	
AN-SL	Yes	Yes	Yes	[40,100]		21 张表和 22 列 2000	

表 2. 查询负载信息

基线方法

1. 我们的基于 GAN 的查询生成框架，记作 GAN。2. 我们的基于 GAN 的查询生成框架，但不编码查询特征信息，记作 GAN-。3. 随机方法通过在我们定义的上下文无关文法中随机选择产生式来生成 SQL 查询，并使用语义规则确保生成的查询在语法上正确。

4. 基于模板的方法使用从给定的查询负载中提取的模板，并随机生成谓词中的常量。

对于每个查询负载，我们使用基线方法生成 2000 个查询。因此，对于每个真实查询负载，我们有四个合成查询负载和一个真实查询负载。

配置 查询生成问题的动作空间受语法限制，因此规模不大。因此，生成器和判别器使用两个轻量级 Transformer，并采用超参数 N=6、H=4 和 d_{model}=128。根据 SeqGAN [18]，，我们使用最大似然法对生成器和判别器进行预训练。为避免生成许多与某些输入查询相同的查询，我们使用 0.3 的丢弃率，并使用 $\lambda = 0.1$ 进行 L_2 正则化，以避免模型过拟合。我们还使用梯度惩罚来避免模式崩溃，同时使生成对抗网络的训练更容易收敛。生成器和判别器的学习率分别为 0.001 和 0.0001。

4.2 生成的查询负载评估

我们首先评估生成的查询负载的质量。为此，我们从多种特征方面比较输入查询负载 Q 与生成的查询负载 Q'，包括字符级相似度、聚合函数的出现次数、运算符的出现次数、嵌套查询的出现次数、连接的表的数量、查询长度、查询基数、由数据库管理系统的查询优化器估算的查询成本，以及属性之间的相关性。

对于某个特定特征 f，设 $f(q)$ 为查询 q的特征 f 的值。在评估中，我们比较所有查询 $q \in Q$ 的 $f(q)$ 分布与所有查询 $q' \in Q'$的 $f(q')$ 分布之间的差异。我们在评估中使用两种度量：

1. Wasserstein 距离（WD）：WD 是受最优质量运输问题启发、用于概率分布的度量。对于定义域相同的两个累积分布函数（cdf） g_1 和 g_2，WD 为

$$W(g_1, g_2) = \int_{\mathbb{R}} |g_1 - g_2| dx$$

其中， x 取自 g_1 和 g_2的定义域。瓦瑟斯坦距离越大，两个分布之间的差异就越大。

2. 最大均值差异（MMD）：最大均值差异也常用于衡量两个分布之间的距离，也可用于高维分布。对于定义域相同的两个分布 Q 和 $Q^{'}$，MMD 表示为：

$$MMD(Q, Q^{'}, F) = ||\frac{1}{n}\sum_{i=1}^{n} F(q_i) - \sum_{j=1}^{m} F(q_j^{'})||_{\mathcal{H}}^2$$

其中，F 是将输入分布映射到希尔伯特空间的核函数。MMD 越大，两个分布之间的差异就越大。

查询负载 方法 序列MMD 基数 WD 成本			WD 长度 瓦瑟斯坦距离		
SPJ-N	GAN	**0.0077**	**0.3665**	**0.3463**	0.843
	GAN-	0.0098	0.6665	0.3573	0.945
	随机方法	0.2895	2.8777	2.1119	5.138
	模板	0.0195	0.6736	0.4673	**0**
AN-N	GAN	**0.0129**	**0.3107**	**0.4629**	1.752
	GAN-	0.0138	0.4357	0.4915	2.3343
	随机方法	0.5847	1.1393	5.2605	29.363
	模板	0.0219	0.5433	0.6973	**0**
AN-S	GAN	**0.0169**	**0.4721**	**0.5498**	2.3374
	GAN-	0.0296	0.5232	0.5979	3.3324
	随机方法	0.5877	2.8232	6.0603	27.9345
	模板	0.0497	0.5467	0.6799	**0**
AN-SL	GAN	**0.0291**	**0.6326**	0.6171	3.876
	GAN-	0.0321	0.6445	0.6312	4.942
	随机方法	0.6571	2.4812	4.0639	22.6485
	模板	0.0513	0.7892	**0.6136**	**0**

表3。查询负载方差评估

实验*1* 表3展示了不同方法生成的查询负载与输入查询负载在若干特征上的分布差异。

序列MDD列展示了生成的查询负载与输入查询负载之间的字符级相似度。可以看出，我们的方法在所有四种查询负载上都取得了最佳结果，其次是模板方法；随机方法生成的查询负载与输入查询负载的差异更大。这是因为我们的方法学习了输入查询负载的分布，并生成与其相似的新查询负载。请注意，对于每种查询负载，完整GAN模型都优于GAN-。这表明，额外的信息——查询负载的成本和基数——能够提升模型的性能。同时，随着查询负载复杂度的增加，尤其是查询长度增加时，序列MMD也会增大。这说明查询负载越复杂，学习其分布就越困难。尽管如此，我们的模型仍将序列MMD保持在较低水平。

基数和成本的瓦瑟斯坦距离（WD）列显示了生成的查询负载与输入查询负载在基数和成本方面的分布差异。在大多数情况下，我们的完整模型效果最佳，其次是 GAN-、模板方法和随机方法。请注意，对于 ANN-N 查询负载，模板方法生成的查询负载在成本分布方面的瓦瑟斯坦距离最小，这

可能是因为对于较长的查询，仅更改查询中的常量不会对查询成本产生显著影响。

长度加权距离列显示了生成的查询负载与输入查询负载在长度分布方面的距离。由于模板方法只更改输入查询负载中的常量，而不改变长度，因此模板方法的长度加权距离为 0。除模板方法外，我们的完整模型在四种查询负载上均取得了最佳结果，其次是 GAN- 和随机方法。

总而言之，生成对抗网络生成的查询负载与输入查询负载更相似，因为我们的方法能更好地学习输入查询负载的分布。此外，模板方法的误差小于随机方法，因为模板是从输入查询负载中提取的，因而部分包含了有关输入查询分布的信息。

实验 *2* 在本实验中，我们考察输入查询负载与生成的查询负载在四个特征上的差异：聚合函数、运算符、嵌套查询以及连接表的数量。结果如图 3 所示。每个子图分为四组，分别表示四种查询负载上的实验结果。在每组中，第一列（记为 GAN）显示使用我们的方法生成的查询负载分布，第二列（记为真实值）显示输入查询负载，第三列（记为模板）显示使用模板方法生成的查询负载，最后一列（记为随机方法）显示使用随机方法生成的查询负载。

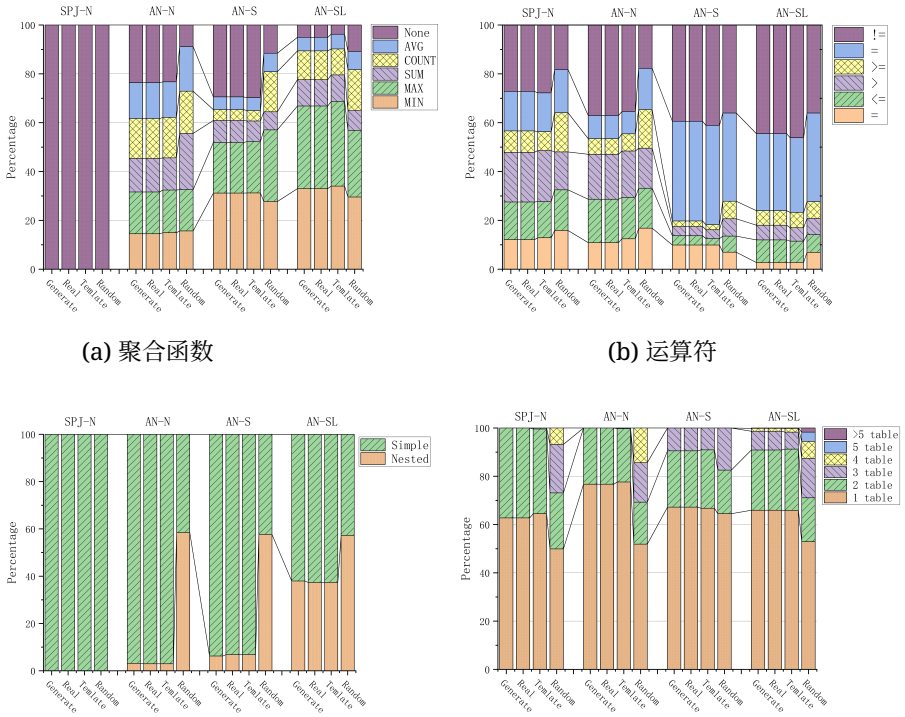
图 3(a) 展示了四种查询负载中聚合函数的分布。由于模板方法不会改变查询结构，因此模板方法生成的查询负载与输入查询负载相同。除模板方法外，我们的方法生成的分布与输入查询负载最为相似，而随机方法的分布最不相似。

请注意，在 AN-N 组中，随机方法生成的查询负载中各聚合函数的占比基本相同，这是因为随机方法选择每种聚合函数的概率相同。然而，在 AN-S 和 AN-LS 组中，AVG 和 SUM 聚合函数的占比较低，因为 AN-S 和 AN-LS 引入了字符串数据，而 AVG 和 SUM 聚合函数不支持字符串。我们的语义模块会限制 AVG 和 SUM 聚合函数的选择。

图 3(b) 展示了运算符的分布。模板方法生成的查询负载与输入查询负载最为相似，其次是我们的方法，最后是随机方法。与聚合函数类似，加入字符串数据后，我们仅支持用于这些数据的 "=" 和 "！ =" 运算符，因此随机方法生成的查询负载中运算符的分布出现了偏斜。

图 3(c) 和 (d) 展示了嵌套查询和连接的表的分布。我们的方法再次优于随机方法。

实验*3* 在本实验中，我们将查询负载中两个属性出现的相关性视为一项重要特征。具体而言，



(a) 聚合函数 (b) 运算符

(c) 嵌套查询 (d) 连接的表

图3. 查询负载中属性出现次数之间的相关性。

对于属性 A，若 A 出现在查询负载的第 i 个查询中，则令 $o_i(A) = 1$；否则，令 $o_i(A) = 0$。然后，对于两个属性 A 和 A'，我们计算 $o_i(A)$ 与 $o_i(A')$ 之间的皮尔逊相关系数。图 4(a) 展示了输入查询负载的相关性热力图，图 4(b) 展示了我们采用生成对抗网络方法生成的查询负载。可以看出，两个属性相关性矩阵非常相似。分别将两个矩阵的所有行向量拼接起来后，我们可以计算这两个拼接向量的余弦相似度，结果为 0.9949。这意味着两个相关性矩阵几乎相同。图 4(c) 展示了采用随机方法生成的查询负载的结果，其相似度为 0.9514。

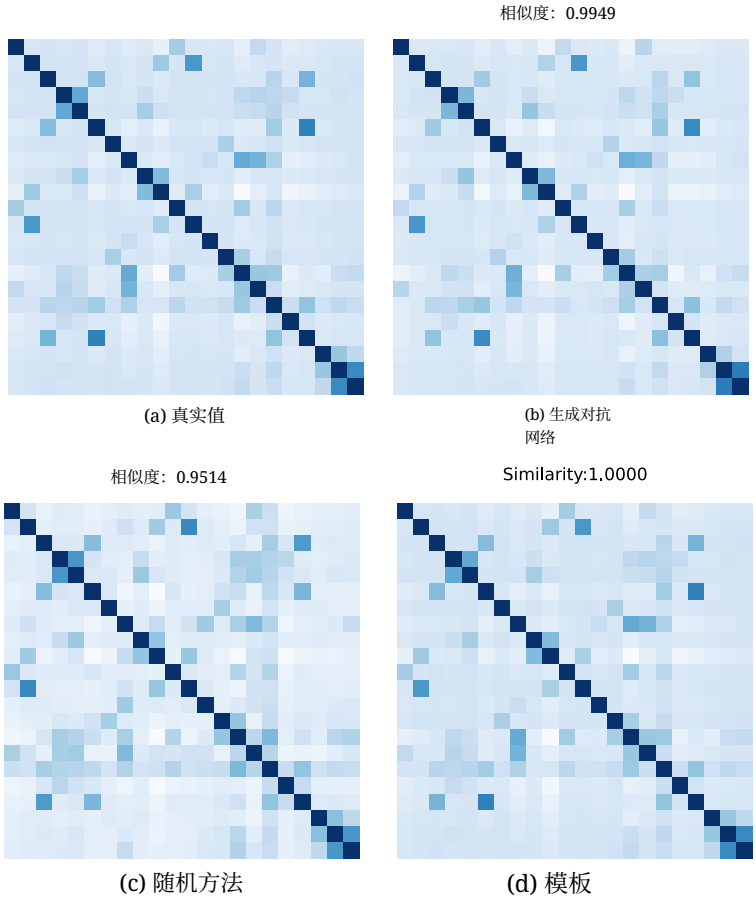


图 4. 热力图。

4.3 下游任务评估

查询生成的一种应用是扩充查询负载。在本实验中，我们采用不同的方法扩充查询负载，然后将扩充后的查询负载用于一项下游任务：基数估计。对于基数估计，我们使用 q-error 指标，其定义如下：

$$Qerror = \frac{1}{n}\sum_{i=1}^{n}\frac{\max\left(y_i, \hat{y_i}\right)}{\min\left(y_i, \hat{y_i}\right)}$$

其中， y_i 表示真实值，$\hat{}y_i$ 表示估计值。

我们选择查询驱动的基数估计器 MSCN [10], ，它使用多重集卷积网络（MSCN）来估计基数。我们使用随机初始化的生成器生成 3000 个查询，并将其输入数据库执行，以获取其真实基数。其中 2000 个查询用作训练集（该训练集较小，需要扩充），1000 个查询用作测试集。

接下来，我们分别基于原始训练集，使用生成对抗网络、模板和随机方法各生成 8000 个查询，将训练集扩充至 10000 个查询。为了与生成方法进行比较，我们还通过复制将原始查询负载扩充至 10,000 个查询。我们使用扩充后的训练集以相同配置训练了四个 MSCN，并使用相同的测试集进行测试。结果如表 4 所示。

	中位数	均值	90th	95th	99th	max
Real	1.307	9.249	6.239	14.072	92.041	2265.9
GAN	1.302	4.269	4.876	9.596	45.291	687.1
模板	1.887	33.145	9.637	21.817	142.483	22432.3
随机方法	3.848	122.084	11.826	39.341	2509.465	16668.8

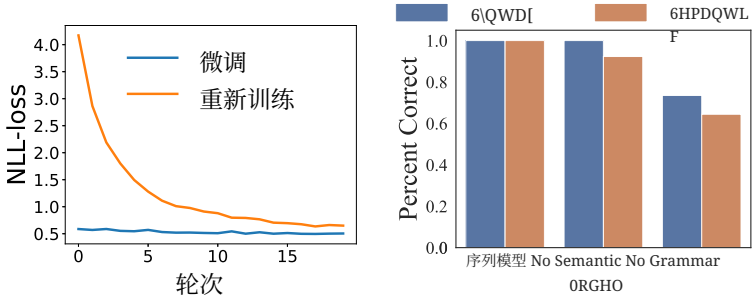
表 4. MSCN 在不同训练集上的 Qerror

总体而言，使用经生成对抗网络扩充后的训练集训练的模型取得了最佳结果。在均值 q误差方面，生成对抗网络方法的准确度是仅使用真实负载训练的模型的 2.6 倍；在最大值 q误差方面，我们的方法的准确度是其 3.3 倍。这是因为我们的模型能够学习输入查询负载的分布，并生成具有相似分布的新查询负载。同时，生成对抗网络的随机性使其能够生成与输入查询略有不同的新查询，从而实现高质量的数据扩充。

使用随机方法和模板方法扩展训练集时，MSCN 的 q误差甚至高于仅使用真实查询负载的情况。我们怀疑，模型使用了大量分布与真实查询负载不同的查询作为训练数据，导致模型拟合了错误的分布，并将错误的查询映射到基数。因此，在测试时，模型无法将真实查询映射到正确的基数。

4.4 我们的方法评估

本节将评估我们提出的查询生成方法中各个组件的有效性。



（a）更新　　（b）句法与语义

图 5. 我们的方法评估。

实验*5* 我们评估了生成对抗网络中句法模块和语义模块的影响。图5(b)展示了生成的查询在句法上正确、语义上正确（或两者都正确）的比例。我们的生成对抗网络模型生成的查询在句法和语义上均完全正确。禁用句法模块后，所有生成的查询在句法上都正确，但只有92.3%在语义上正确。当句法模块和语义模块都被禁用时，生成的查询中只有64.4%在语义上正确。

实验*6* 我们还测试了查询负载发生变化时模型的更新能力。我们采用两种更新策略：重新训练和微调。微调是指基于当前模型，使用新的查询负载继续训练。为比较这两种更新策略的差异，我们合成了两个查询负载 A 和 B，每个负载包含 2000 个查询，查询长度范围分别为 $[0,40]$ 和 $[40,100]$。结果如图 5(a) 所示。我们首先使用负载 A 训练一个生成对抗网络模型，然后采用两种策略针对负载 B 进行更新。与重新训练相比，微调策略的收敛速度要快得多。微调在 3 个训练轮次内就达到了较低的负对数似然损失，而重新训练则需要 15 个训练轮次。这可能是因为预训练模型保留了关于查询的一些基本信息。当查询负载略有变化时，微调可以帮助模型快速学习新的查询负载分布。

5 相关工作

现有的查询生成方法大致可分为三类：随机方法、基于模板的方法和基于强化学习的生成。

随机方法 随机方法生成的查询负载主要用于数据库的大规模随机错误诊断和测试。典型方法包括 RAGS [15] 和 SQL-SMITH [1]。RAGS 通过遍历随机解析树并将其打印出来来生成 SQL语句。SQL-SMITH 生成随机查询抽象语法树（AST），然后使用格式化工具将其转换为 SQL 查询。尽管随机方法生成的查询符合 SQL语法，但它们完全是随机的。即使是 SQL-SMITH 也可能生成语义不正确的查询，导致查询执行失败或执行结果为空。

基于模板的方法 基于模板的方法使用给定的查询模板，并调整谓词中的值，以生成满足特定约束的查询。Bruno 等人 [3] 使用启发式爬山算法，在谓词参数空间中搜索，以生成满足基数约束的查询。Chaitanya 等人 [13] 将 d维谓词空间平均划分为 k^d 个较小单元，在 d维空间中进行单元级搜索，并选择得分最高的 k 个单元重新划分并搜索。尽管基于模板的方法能够生成满足特定约束的查询，但仍无法生成与给定查询负载具有相似分布的新查询。

基于强化学习的方法 基于强化学习（RL）的查询生成方法无需提供查询即可生成满足特定约束的查询。典型方法之一是 LearnedSQLGen [20]。LearnedSQLGen 采用演员-评论家强化学习网络结构，并在每个时间步选择合适的词元来生成查询。此外，为了保证生成的查询在语法和语义上的正确性，LearnedSQLGen 使用有限状态机来限制动作空间。然而，LearnedSQLGen 无法捕捉给定查询负载的分布。

另一类相关研究是文本序列生成，这是自然语言处理（NLP）中一项得到广泛研究的技术。查询生成本质上是一个文本序列生成问题，因为查询以 SQL语句的形式表达。长短期记忆网络（LSTM） [7] 和循环神经网络（RNN）在文本生成方面表现出色。训练循环神经网络最常用的方法是最大似然法 [14]。然而，Bengio 等人 [2] 指出，最大似然法会在推理阶段造成暴露偏差。SeqGAN [18]将生成对抗网络结构引入文本生成，并使用强化学习中的蒙特卡罗方法训练生成器，从而解决了暴露偏差问题。

许多序列生成方法是完全随机的过程 [7,18]，会逐步从词元空间中选择合适的词元。以这种方式生成的查询可能会出现语法错误。例如，我们正在生成一个 SQL 查询，而生成器已经生成了“SELECT *”。下一步，生成器从词元空间中选择 *，导致产生不正确的查询“SELECT **”。即使我们保证查询在句法上正确，也可能出现语义错误。在 SQL 中，常见的语义错误包括操作数类型不匹配、无意义的谓词表达式以及不恰当的表连接。语义错误可能导致查询结果为空、产生非预期的查询结果，甚至导致查询无法执行。TreeGAN [12] 将上下文无关文法引入序列生成过程，从而能够生成符合语法的文本。基于深度学习的文本生成方法为研究我们的查询生成方法提供了技术基础。

6 结论

我们提出了一种基于生成对抗网络（GAN）的查询生成方法。GAN 模型能够很好地捕捉输入查询的各种特征分布。此外，我们采用上下文无关文法和语义规则来指导查询生成，从而确保生成的查询在句法和语义上均正确。实验验证了我们的方法能够生成与输入查询负载非常相似的新查询负载。

参考文献

1. A.Seltenreich：Sqlsimith（2020），https://github.com/anse1/sqlsmith 2. Bengio, S., Vinyals, O., Jaitly, N., Shazeer, N.：使用循环神经网络进行序列预测的计划采样。《神经信息处理系统进展》**28** （2015）3. Bruno, N., Chaudhuri, S., Thomas, D.：生成满足基数约束的查询以进行数据库管理系统测试。《IEEE知识与数据工程汇刊》**18**（12），1721–1725（2006）4. Curino, C., Jones, E.P.C., Zhang, Y., Madden, S.R.：Schism：一种由工作负载驱动的数据库复制与分区方法（2010）5. Dutt, A., Wang, C., Nazi, A., Kandula, S., Narasayya, V., Chaudhuri, S.：使用轻量级模型估计范围谓词的选择率。《VLDB基金会会刊》**12**（9），1044–1057（2019）6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.：生成对抗网络。《ACM通讯》**63**（11），139–144（2020）7. Hochreiter, S., Schmidhuber, J.：长短期记忆。《神经计算》**9**（8），1735–1780（1997）8. Karras, T., Laine, S., Aila, T.：用于生成对抗网络的基于风格的生成器架构。载于：IEEE/CVF计算机视觉与模式识别会议论文集。第4401–4410页（2019）9. Kingma, D.P., Welling, M.：自编码变分贝叶斯。arXiv预印本 arXiv:1312.6114（2013）

10. Kipf, A., Kipf, T., Radke, B., Leis, V., Boncz, P., Kemper, A.：学习型基数估计：使用深度学习估计相关连接。arXiv预印本 arXiv:1809.00677（2018）11. Li, G., Zhou, X., Li, S., Gao, B.：Qtune：一种基于深度强化学习的查询感知数据库调优系统。《VLDB基金会会刊》**12**（12），2118–2130（2019）12. Liu, X., Kong, X., Liu, L., Chiang, K.：TreeGAN：使用生成对抗网络进行语法感知序列生成。载于：2018年IEEE国际数据挖掘会议（ICDM）。第1140–1145页。IEEE（2018）13. Mishra, C., Koudas, N., Zuzarte, C.：为数据库测试生成目标查询。载于：2008年ACM SIGMOD国际数据管理会议论文集。第499–510页（2008）14. Salakhutdinov, R.：学习深度生成模型。《统计学及其应用年度评论》**2**，361–385（2015）15. Slutz, D.R.：SQL大规模随机测试。载于：VLDB。第98卷，第618–622页。Citeseer（1998）16. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.：注意力就是你所需要的一切。《神经信息处理系统进展》**30**（2017）17. Yin, P., Neubig, G.：一种通用代码生成的句法神经模型。arXiv预印本 arXiv:1704.01696（2017）18. Yu, L., Zhang, W., Wang, J., Yu, Y.：SeqGAN：使用策略梯度的序列生成对抗网络。载于：AAAI人工智能大会论文集。第31卷（2017）19. Yuan, H., Li, G., Feng, L., Sun, J., Han, Y.：结合深度学习与强化学习的自动视图生成。载于：2020年IEEE第36届国际数据工程会议（ICDE）。第1501–1512页。IEEE（2020）20. Zhang, L., Chai, C., Zhou, X., Li, G.：LearnedSQLGen：使用强化学习进行约束感知SQL生成。载于：2022年国际数据管理会议论文集。第945–958页（2022）21. Zhong, R., Chen, Y., Hu, H., Zhang, H., Lee, W., Wu, D.：Squirrel：利用语言有效性与覆盖率反馈测试数据库管理系统。载于：2020年ACM SIGSAC计算机与通信安全会议论文集。第955–970页（2020）