Методы автоматизированной генерации программного кода по тексту на естественном языке

Автор: Шпарута Софья, студент 1-го курса магистратуры математико-механического факультета Санкт-Петербургского Государственного Университета

Научный руководитель: Графеева Наталья Генриховна, кандидат физико-математических наук, доцент кафедры ИАС

Введение

В настоящее время множество приложений имеют естественно-языковой интерфейс. Алиса, Google-поиск, Яндекс-карты и многие другие программы позволяют нам «общаться» с ними на естественном для нас, людей, языке.

Поэтому, задача перевода из естественного языка в программный код очень актуальна в наши дни. Сейчас существует очень много различных статей и решений на эту тему. В частности, естественно-языковые интерфейсы к реляционным базам данным привлекают довольно много внимания.

Хотелось бы создать систему, которая транслировала бы вопросы с естественного языка в SQL-запрос. Это позволило бы очень многим людям, не владеющим техническими знаниями, делать различные действия с базами данных. В дополнение, большое удобство принесла бы быстрая скорость подобной системы.

В данной работе представлены

- Обзор существующих решений;
- Описание базы данных достопримечательностей Санкт-Петербурга;
- Описание приложения, транслирующего вопрос с естественного языка в SQL-запрос и которое выдает ответ.

Обзор существующих решений

Генерация SQL-запросов с использованием синтаксических зависимостей и метаданных на естественном языке

Введение. Авторы статьи **[6]** решают проблему перевода вопроса на естественном языке во что-то понятное машиной автоматическим способом, генерируя SQL-запросы, структура и компоненты которых соответствуют концепциям NL (выраженным как слова) и грамматическим зависимостям. Их новая идея заключается в том, как и где это сопоставление можно найти.

Определение проблемы. Авторы утверждают, что можно выполнить несколько SQL-запросов через информационную схему (IS) и целевую базу данных и объединить их результирующие наборы, чтобы генерировать SQL-запрос простым и интуитивно понятным способом.

Кроме того, можно выполнить вопрос, отвечающий за несколько баз данных, поскольку информационная схема хранит метаданные всех баз данных, которые находятся в одной машине.

Еще одна важная особенность, которая должна быть принята во внимание – потенциальная сложность вопроса NL (подчиненные, конъюнкция, отрицание), которые могут иметь собственное соответствие во вложенном SQL-запросе. Общий SQL-запрос, который может обрабатывать рассматриваемая система, имеет следующий вид:

SELECT DISTINCT COLUMN FROM TABLE [WHERE УСЛОВИЕ]

Авторы нашли алгоритм сопоставления, который соответствует зависимостям между компонентами NL и структурой SQL, что позволяет построить набор возможных запросов, которые отвечают на заданный вопрос. Чтобы представить текстовые отношения предложения NL, они используют типизированные отношения зависимости.

Проблема построения SQL-запроса Q, который получает ответ для заданного вопроса NL, сводится к следующей проблеме. Заданный вопрос q, представленный с помощью его типизированных зависимостей, сгенерировал три набора предложений S; F; W таких, что множество A = SELECT S x FROM x WHERE W содержит все возможные ответы на q и выбирает тот, который

максимизирует вероятность правильного ответа на вопрос Q.

Наборы для построения. Первый шаг до создания всех возможных запросов для вопроса q заключается в создании их компонентов S; F; W, то есть SELECT, FROM и WHERE, начиная с автоматической генерации SQL-запросов из списка зависимостей SDC_q. Этот список должен быть:

а) предварительно обработан с использованием обрезки, сгенерирования и добавления синонимов;

б) проанализирован для создания набора стеблей, используемых для сборки S и W;

с) изменен/очищен, чтобы сохранить зависимость, используемую на конечном рекурсивном этапе для генерации вложенных запросов.

Сначала обрезаются те отношения, которые бесполезны для обработки, чтобы получить оптимизированный список SDC_q^{opt}.

Затем для каждого грамматического отношения в SDC_q^{opt} авторы применяют итерационный алгоритм, который добавляет эти стемы соответственно к проекции и/или селекции категорий, соответствующих установленным правилам, которые из-за нехватки места не могут быть перечислены здесь. Однако ключевая идея состоит в том, чтобы использовать ориентированные на проекцию отношения и ориентированные на выбор рекурсивные категоризации стеблей.

Затем используется проекция для поиска в метаданных всех полей, которые могут совпадать с ориентированными на проекцию стемами. На основании того, сколько совпадений найдено, вес w определяется для каждой проекции, получая предложение SELECT замыкания множества S.

Вместо этого, ориентированный на селекцию набор стемов следует разделить на два разных набора стемов – правый и левый. Набор L содержит стебли, которые соответствуют их сопоставлению в IS, тогда как для остальных стеблей – R – нужно искать в базе данных для сопоставления. Для построения предложений WHERE сначала генерируются базовые выражения expr = eL OP eR, а затем объединяются посредством конъюнкции и отрицания, сохраняя только те выражения expr, что выполнение проекции от селекции не приводит к ошибке, по крайней мере, для таблицы в базе данных. Это не значит, что R может быть пустым множеством, например, когда условие WHERE требует гнездования; в этом случае eR будет весь подзапрос. Более того, также L может быть пустым. Это неудивительно, поскольку в SQL-запросе предложение WHERE не является обязательным. Однако отсутствие ориентированных на выбор стеблей не обязательно означает, что W должно быть пустым.

Наконец, после того, как были построены два множества S и W, генерация предложения FROM F проста. Этот набор должен содержать только все таблицы, к которым относятся предложения в S и W, обогащенные парными объединениями

Создание запросов. Отправной точкой для ответа на запрос является генерация множества A = {SxFxW} ∪ {SxF}. Если такой запрос существует, должна существовать тройка <s, f, w> ∈ A, такая что выполнение SELECT s FROM f [WHERE w] возвращает правильный ответ. На этом этапе множество A содержит все допустимые кортежи, среди которых еще есть кто-то не полезный. Одним из примеров являются бессмысленные запросы: те, которые проектируют одно и то же поле по сравнению со значением в выборе.

В частности, существуют избыточные запросы, которые после оптимизации могут привести к дублированию в наборе; следовательно, его мощность ниже, чем в теории.

Авторы добавляют вес к каждому предложению в S и W. Этот вес является простой мерой, состоящей в подсчете количества стемов, возникших в статье. При суммировании предложений совокупный вес просто вычисляется как сумма его компонентов и используется для упорядочения полученного набора $\overline{A}$ возможных полезных запросов от наиболее вероятного до менее одного.

Стоит отметить, что может быть больше запросов с одинаковым весом. Чтобы справиться с этим, запросам предоставляются привилегии, которые связаны с меньшими объединениями, и теми, которые встраивают наиболее значимую (например, ссылочную) таблицу.

SQLNet: Генерация структурированных запросов по вопросам на естественном языке без усиленного обучения. **[7]**

Введение. Фактически стандартный подход для решения проблемы семантического

разбора заключается в том, чтобы рассматривать как описание естественного языка, так и SQL-запрос в качестве последовательностей и обучать модель S2S или ее варианты, которые могут использоваться как синтаксический анализатор. Одна из таких проблем заключается в том, что разные SQL-запросы могут быть эквивалентны друг другу из-за коммутативности и ассоциативности. Например, рассмотрим следующие два запроса:

SELECT result WHERE goal=16 AND score='1-0'

SELECT result WHERE score='1-0' AND goal=16

Порядок двух ограничений в предложении WHERE не влияет на результаты выполнения запроса, но синтаксически они рассматриваются как разные запросы. Хорошо известно, что порядок этих ограничений влияет на производительность модели с последовательностью в последовательности, и, как правило, трудно найти лучший порядок.

В этой работе авторы предлагают SQLNet принципиально решить эту проблему, избегая структуры S2S, когда порядок не имеет значения. В частности, используется подход на основе эскиза для генерации SQL-запроса. Эскиз естественно ориентируется на синтаксическую структуру SQL-запроса. Нейронная сеть, называемая SQLNet, используется для прогнозирования содержимого для каждого слота в эскизе.

Как обсуждалось выше, наиболее сложной задачей является генерация предложения WHERE. По сути, проблема с декодером S2S заключается в том, что предсказание следующего токена зависит от всех ранее созданных токенов. Однако разные ограничения могут не иметь зависимости друг от друга. В данном подходе SQLNet использует эскиз, чтобы обеспечить зависимость между разными слотами, чтобы предсказание для каждого слота основывалось только на предсказаниях других слотов, от которых оно зависит. Для реализации этой идеи дизайн SQLNet вводит две новые конструкции: последовательности и столбцы. Первая предназначена для прогнозирования неупорядоченного набора ограничений вместо упорядоченной последовательности, а вторая предназначена для захвата отношения зависимостей, определенных в эскизе при прогнозировании.

Авторы оценивают свой подход по набору данных WikiSQL, который, насколько нам известно, является единственным крупномасштабным набором данных NL2SQL и сравнивается с самым современным подходом Seq2SQL. Указанный подход приводит к высокой точности совпадения запросов и точности результата на наборе тестов WikiSQL. Другими словами, SQLNet может добиться высокой точности совпадения запросов и результатов запроса.

Первоначально предлагалось установить набор данных WikiSQL для обеспечения того, чтобы набор обучения и тестовый набор не пересекались. В практических условиях более вероятно, что такое решение NL2SQL будет развернуто там, где существует хотя бы один запрос, наблюдаемый в обучающем наборе для большинства таблиц.

Синтез SQL запроса, полученных из вопросов на естественном языке и табличных схем. Пусть входные данные содержат две части: вопрос на естественном языке, содержащий запрос для таблицы, и схему запрашиваемой таблицы. Схема таблицы содержит как имя, так и тип каждого столбца. Вывод представляет собой SQL-запрос, который отражает вопрос о естественном языке в отношении запрошенной таблицы.

Обратите внимание, что задача WikiSQL рассматривает синтез SQL-запроса только для одной таблицы. Таким образом, в выходном SQL-запросе необходимо предсказать только предложение SELECT и предложение WHERE, и предложение FROM может быть опущено.

Свойства WikiSQL

Во-первых, он предоставляет крупномасштабный набор данных, чтобы можно было эффективно обучать нейронную сеть.

Во-вторых, он использует источники толпы для сбора вопросов естественного языка, созданных людьми, так что он может помочь преодолеть проблему, которую хорошо обученная модель может переделать в шаблонные описания.

В-третьих, нас интересует проблема синтеза SQL в настройках предприятия. В такой настройке база данных может содержать миллиарды конфиденциальной информации пользователей, и, таким образом, как справляться с масштабируемостью данных и как обеспечить конфиденциальность являются важными проблемами. Поэтому мы авторы предпочли

решение, которое синтезирует SQL-запросы только из описания естественного языка и схемы таблицы.

В-четвертых, данные разделяются так, что тренировочный и тестовый набор не разделяют таблицы. Это помогает оценить способность подхода обобщаться на невидимую схему. Предыдущие наборы данных могут иметь одно или несколько из таких четырех свойств, но, насколько нам известно, мы не знаем ни одного набора данных NL2SQL, имеющего все эти свойства, кроме WikiSQL.

Далее авторы рассматривают вопрос о создании и решении задачи синтеза SQL более сложных запросов как важной будущей работы.

Что представляет собой SQLNet. В отличие от существующих моделей семантического анализа, которые разработаны для агностики выходных грамматик, основная идея статьи состоит в том, чтобы использовать эскиз, который очень хорошо согласуется с грамматикой SQL.

Поэтому SQLNet необходимо заполнить слоты в эскизе, а не предсказывать как выходную грамматику, так и контент.

Эскиз разработан так, чтобы быть достаточно общим, чтобы все интересующие SQL запросы могли быть выражены эскизом. Поэтому использование эскиза не препятствует обобщаемости нашего подхода.

Эскиз отражает зависимость прогнозов. Таким образом, предсказание значения одного слота зависит только от значений этих слотов, от которых оно зависит. Это позволяет избежать проблемы «вопросов порядка» в модели последовательности к последовательности, в которой одно предсказание обусловлено всеми предыдущими предсказаниями. Чтобы делать прогнозы на основе эскиза, мы разрабатываем два метода: последовательность для набора и внимание к колонке.

Детали модели и подготовки данных SQLNet. В этом разделе представлена полная модель SQLNet и детали обучения. Предсказания предложения SELECT и предложения WHERE разделены.

Предсказание предложения WHERE. Предложение WHERE является самой сложной структурой для прогнозирования в задаче WikiSQL. Указанная модель SQLNet сначала предсказывает набор столбцов, которые появляются в предложении WHERE, а затем для каждого столбца он генерирует ограничение, предсказывая интервалы OP и VALUE, которые будут описаны ниже.

Слоты колонок. После вычисления вероятностей для каждого столбца при данном запросе SQLNet необходимо решить, какие столбцы включить в WHERE. Один из подходов – установить порог. Однако можно обнаружить, что альтернативный подход обычно может дать лучшую производительность. В частности, используется сеть, чтобы предсказать общее число столбцов K, которые должны быть включены в подмножество, и выбираются столбцы с наибольшей вероятностью, чтобы сформировать имена столбцов в предложении WHERE. Видно, что большинство запросов имеют ограниченное число столбцов в своих предложениях WHERE. Поэтому устанавливается верхняя граница на количество выбранных столбцов, и поэтому авторы ставят задачу предсказать количество столбцов как проблему классификации (N +1) - уровней (от 0 до N). В частности, имеется N = 4, чтобы упростить настройку оценки. Но нужно учесть, что ы можно избавиться от гиперпараметра N, используя модель предсказания с вариантной длиной, такую как модель предсказания столбца SELECT.

Заключение. В конце авторы проводят подробное сравнение и оценку работы SQLNet и модели S2S. Делаются выводы, что все существующие подходы, использующие модель S2S, страдают от проблемы «порядка», когда порядок не имеет значения. Предыдущие попытки использования обучения с подкреплением для решения этой проблемы приносят лишь небольшое улучшение, например, примерно на 2 пункта. В данной работе SQLNet принципиально решает проблему «порядка вещей», используя модель S2S для создания SQL-запросов, когда заказ не имеет значения.

SQLizer: Синхронизация запросов с естественного языка **[9]**

Введение. Существующие методы автоматического синтеза SQL-запросов подразделяются на два разных класса: подходы, основанные на шаблонах и те, которые основаны на естественном языке.

Методы программирования по шаблону требуют от пользователя представления миниатюрной версии базы данных вместе с

ожидаемым выходом. Недостатком методов, ориентированных на шаблоны, является то, что они требуют, чтобы пользователь был знаком с схемой базы данных. Более того, поскольку реалистичные базы данных обычно включают в себя множество таблиц, пользователю может быть довольно сложно выразить свое намерение с использованием примеров ввода-вывода.

С другой стороны, методы, которые могут генерировать SQL-запросы из описания естественного языка (NL), легче для пользователей, но сложнее для базового синтезатора, поскольку естественный язык по своей сути неоднозначен. Существующие методы на основе NL стараются достичь высокой точности путем обучения системы конкретной базе данных. Следовательно, существующие методы NL для синтеза SQL-запросов либо не полностью автоматические, либо требуют настройки для каждой базы данных.

В этой статье авторы представляют новую технику и ее реализацию в инструменте под названием Sqlizer для синтеза SQL-запросов из описаний на английском языке. Данный метод полностью автоматизирован и требует обучения или настройки под конкретную базу данных.

Обзор. На рисунке 1 показана соответствующая часть схемы для базы данных Microsoft Academic Search (MAS), и

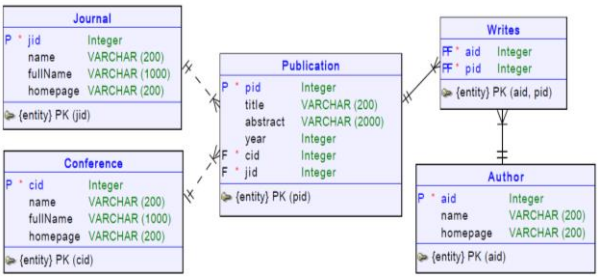


Рисунок 1. Схема для базы данных Microsoft Academic Search

предположим, что мы хотели бы синтезировать запрос базы данных для получения количества документов в OOPSLA 2010. Чтобы использовать этот инструмент, пользователь предоставляет описание на английском языке, например, «Find the number of papers in OOPSLA 2010». Далее изложены шаги, предпринятые Sqlizer при синтезе желаемого SQL-запроса.

Генерация эскизов. В данном подходе сначала используется семантический синтаксический анализатор для генерации лучших k наиболее вероятных программных эскизов. В этом примере эскиз запроса с наивысшим рангом, возвращаемый семантическим парсером, следующий:

SELECT count(?[papers]) FROM ??[papers] WHERE ? = "OOPSLA 2010"

‘??’ представляет собой неизвестную таблицу, а ‘?’ - неизвестные столбцы. В случае присутствия, слова, записанные в квадратных скобках, представляют собой так называемые «подсказки» для соответствующих пропусков. Например, вторая подсказка в этом эскизе указывает, что таблица, представленная символом ‘?’, семантически похожа на английское слово «papers».

Первая итерация. Начиная с приведенного выше эскиза S, Sqlizer перечисляет все хорошо типизированные пополнения q_i из S вместе со счетчиком для каждого q_i. В этом случае существует много возможных типизированных доработок S, однако ни один из них не соответствует нашему доверительному порогу. Например, одна из причин, почему Sqlizer не находит запрос с высокой степенью уверенности, заключается в том, что в любой из таблиц базы данных нет записи под названием «OOPSLA 2010».

Затем Sqlizer выполняет локализацию ошибок на S, чтобы определить основную причину сбоя (т. е. Не соответствует порогу доверия). В этом случае мы определяем, что вероятной основной причиной является предикат ‘?’ = «OOPSLA 2010», так как нет записи в базе данных «OOPSLA 2010», и алгоритм синтеза присвоил низкий доверительный балл до этого срока. Затем эскиз восстанавливается, разделяя предложение where на два отдельных конъюнкта. В результате получаем следующее уточнение S' исходного эскиза S:

SELECT count(?[papers]) FROM ??[papers] WHERE ? = "OOPSLA" AND ? = 2010

Вторая итерация. Затем Sqlizer пытается завершить очищенный эскиз S', но он снова не может найти завершение с высоким доверием S'. В этом случае проблема заключается в том, что нет единой таблицы базы данных, содержащей как запись «OOPSLA», так и запись «2010». Возвращаясь к локализации ошибок, алгоритм теперь определяет, что наиболее вероятной проблемой является термин [papers], и пытается восстановить его, введя соединение. В результате новый эскиз S" теперь становится следующим:

SELECT count(?[papers]) FROM ??[papers] JOIN ?? ON ? = ? WHERE ? = "OOPSLA" AND ? = 2010

Третья итерация. Вернувшись на этап завершения эскиза в третий раз, мы теперь можем найти инстанцирование q с уверенностью q. В этом случае наивысшее ранжированное завершение S" соответствует следующему запросу:

SELECT count(Publication.pid) FROM Publication JOIN Conference ON Publication.cid = Conference.cid WHERE Conference.name = "OOPSLA" AND Publication.year = 2010

Этот запрос действительно правильный, и его запуск в базе данных MAS дает количество документов в OOPSLA 2010.

Общая методика синтеза

Algorithm 1 General synthesis methodology

```
1:  procedure SYNTHESIZE(Q, Γ, γ)
2:      Input: natural language query Q, type environment Γ, confidence threshold γ
3:      Output: A list of synthesized programs and their corresponding confidence scores
4:      Sketches := SEMANTICPARSE(Q)                          ▷ Sketch generation
5:      Programs := [ ];
6:      for all top k ranked S ∈ Sketches do
7:          loop n times
8:              θ := FINDINHABITANTS (S, Γ)                   ▷ Type-directed sketch completion
9:              needRepair := true
10:             for all (I_i, P_i) ∈ θ do
11:                 if P_i > γ then Programs.add(I_i, P_i); needRepair := false
12:             if ¬needRepair then break
13:             F := FAULTLOCALIZE(S, Γ, ∅)                   ▷ Sketch refinement
14:             if F = null then break
15:             S := S[FINDREPAIR(F)/F]
16:     return Programs
```

Заключение. Авторы предложили новую методологию синтеза программ с естественного языка и применили ее к проблеме синтеза кода SQL из английских запросов. Начиная с первоначального эскиза программы, сгенерированного с использованием семантического разбора, этот подход входит в цикл итеративного уточнения, который чередуется между типичным типом обитания и ремонтом эскиза. В частности, метод использует знания, специфичные для домена, для присвоения оценок доверия типам (например, эскиза) жителей и использует эти оценки доверия для указания локализации ошибок. Неисправные субтермы, определяемые с помощью локализации ошибок, затем восстанавливаются с использованием базы данных о тактике восстановления домена.

Авторы реализовали предложенный подход в инструменте под названием Sqlizer, сквозной системе для генерации SQL-запросов с естественного языка. Их эксперименты по 455 запросам из трех разных баз данных показывают, что Sqlizer оценивает желаемый запрос как верхний в 78% случаев и среди 5 лучших в ~ 90% случаев.

Генерация SQL-запросов из естественного языка **[10]**

Подход авторов основывается на глубокой нейронной сети, которая переводит вопросы на естественном языке в SQL-запросы. SQL-запрос может быть разбит на 3 части: SELECT, operator и WHERE. Идея заключается в том, чтобы избежать S2S подхода, где порядок не имеет значения.

Авторы выбрали датасет WikiSQL, представленную в одной из предыдущих работ.

Они тренируют компоненты селекции и агрегации независимо используя потерю кросс-энтропии.

Представленная модель схожа с SQLNet, но имеет свои особенности.

Пока данная работа ограничена запросами вида: select…from…where, но вполне может быть расширена до join.

Независимая архитектура схемы БД для перевода из NL в SQL **[11]**

Введение. На протяжении многих лет программисты пытаются преодолеть языковой барьер между пользователем и компьютером. В большинстве мест компьютер используется для сортировки данных и поиска (в соответствии с потребностями и требованиями пользователя). В течение последних десятилетий в преобразовании NL в SQL существует много работ. Процесс преобразования NL в SQL разделен на пошаговые уровни. Например, морфология имеет дело с наименьшей частью слова. Лексический уровень имеет дело с структурой предложения и символикой. Здесь также рассмотрены возможные значения и выбирается то, которое подходит.

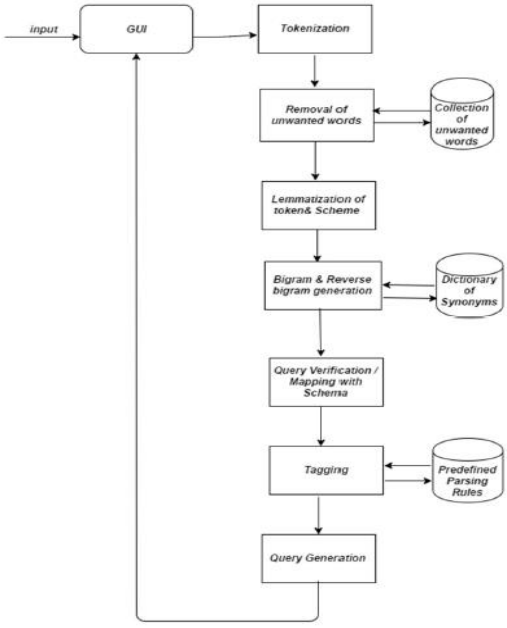
Объем предлагаемой системы. В этой статье мы предлагаем общую систему для преобразования NL SQL. Во-первых, мы представляем область применения системы для лучшего понимания предлагаемой архитектуры системы, которая представлена ниже. Поскольку мы знаем, что сфера охвата

системы очень важна для точного понимания работы системы. Наша система будет работать лучше всего, выполняя следующие ограничения / область действия w.r.t. базы данных и ее схемы:

1. Система принимает только запрос на английский язык.
2. Названия таблиц и имена столбцов должны быть полными английскими словами в схеме базы данных. Это не должно быть сокращением или сокращенным словом, поскольку это снижает эффективность и точность предлагаемой системы.
3. Если любое имя сущности схемы является многословным, оно должно быть связано с переходом.
4. Имя многословного слова должно содержать максимум два слова.
5. Имя столбца должно быть уникальным в схеме.
6. Система может принимать только однострочный запрос, который может быть расширен в будущем.

Архитектура системы

Архитектура предлагаемой системы



состоит из следующих шагов:

- Токенизация
- Удаление нежелательных слов
- Лемматизация токенов запросов, а также объектов схемы
- Генерация слов двухбайтовых объектов и их обратная комбинация
- Проверка запроса / Сопоставление с помощью схемы
- Маркировка
- Генерация запроса

Данные этапы очень схожи с тми, которые использовались в одноименной ВКР.

Заключение. В этой статье показано, что концепция преобразования запросов NL к SQL может поддерживать несколько схем баз данных.

Выбор базы данных

На рисунке представлена база данных достопримечательностей Санкт-Петербурга. Она составлена вручную. Сейчас содержится около 20 записей. В данный скрипты для её создания находятся в открытом доступе по ссылке: https://github.com/shparutask/TranslationSystem/tree/master/Scripts/SPB

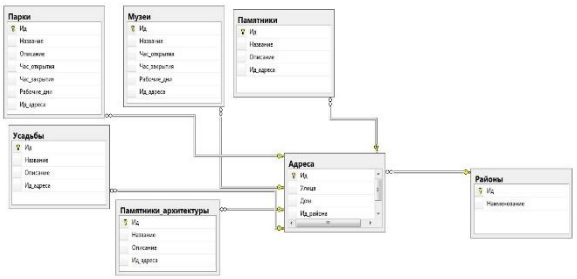


Рисунок 2. Схема для базы данных достопримечательностей Санкт-Петербурга

Базовая реализация

Данная работа реализована в виде пользовательского приложения для Windows на языке C# с использованием функций из работы **[5]**. Приложения содержит в себе 3 проекта: UI, библиотека трансляции, и тестовый проект. Библиотека трансляции представляет собой dll-библиотеку на C#.

Для построения лексикона заведен класс dbGraph, котором хранится информация о БД, таблицах и связях между ними. Также он помогает уловить связи таблиц на этапе 4 (см. пред. раздел). В конструкторе прописано построение самого графа базы данных. На Рисунке 3 ниже можно увидеть строение узла графа – класс Table.

Table
- _fK : Dictionary<string, ForeignKey> + Columns : string[] + FK : Dictionary<string, ForeignKey> + Name : string
+ Table(string[] columns, string name)

Рисунок 3. Описание класса Table

С помощью экземпляра класса *SqlConnection* происходит подключение к БД и выгрузка имен таблиц, названий их колонок и информации о внешних ключах. Затем вся эта информация заносится в граф – массив экземпляров класса *Table*. Также класс *dbGraph* содержит метод *ReturnDataValues*, который по названию таблицы возвращает коллекцию меток в формате (значение, “таблица.колонка”). В данном приложении заведен класс *Lexicon*. Описание на Рисунке 4.

Lexicon
+ value_tags : List<ValueTag> - language_tags : List<string>[] - lx : List<TaggedWord>
+ Lexicon(dbGraph g) + Add(TaggedWord word) + getAll() : List<TaggedWord> + changeLx(List<TaggedWord> new_wlist)

Рисунок 4. Описание класса Lexicon

В этом классе хранится коллекция слов, помеченная тремя видами тэгов: тег-группа, тег-метка и POS-тэг. Тэг-группа имеет 2 вида: Значение(Value) и Объект (Object). Языковые тэги (language-tags) загружаются также с помощью dbGraph. Таким образом реализован 1 этап.

Для грамматики заведен класс Grammar (см. Рис. 5), который содержит в себе правила и методы, для построения дерева парсинга.

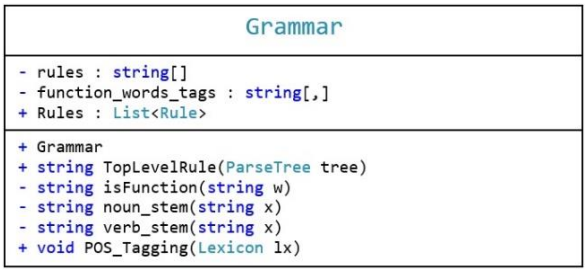


Рисунок 5. Описание класса Grammar

Сначала, каждому слову приписываются, так называемые POS-тэги в методе *POS_Tagging*, которые показывают принадлежность определенной части речи. После чего из них строится узел *ParseNode*. В итоге получается, так называемое, дерево парсинга, которое представлено классом *ParseTree* (см. Рисунок 6), которое для

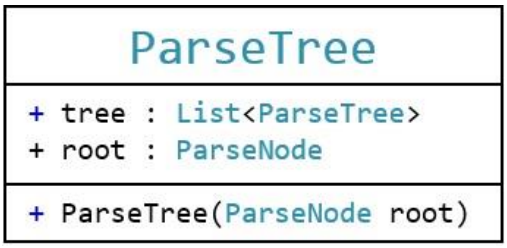


Рисунок 6. Описание класса Parse Tree

построения непосредственно дерева использует *ParseTreeBuilder* (см. Рисунок 7).

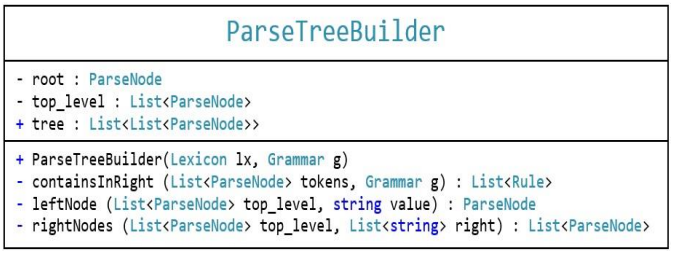


Рисунок 7: Описание класса ParseTreeBuilder

Все этапы из предыдущего раздела собраны в одном классе Translation (Рисунок 8), который на вход получает вопрос и поэтапно обрабатывает его.

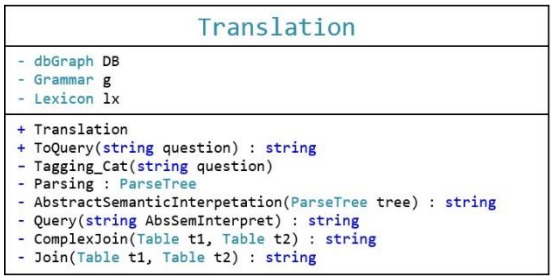


Рисунок 8. Описание класса Translation

Доработки реализации

Описанная ранее реализация позволяет перевести именно с английского языка на язык SQL. В данный момент стоит задача перевода с русского языка. Для этого к существующей реализации был добавлен класс YandexTranslator,

КАРТИНКА

который содержит в себе API компании Yandex для перевода с русского языка на английский, а также класс Tesaurus,

КАРТИНКА

который представляет собой пополняемое хранилище – аналог русско-английского словаря.

Результат

В результате работы реализовано приложение, в котором в поле ввода вводится вопрос, затем, по нажатии кнопки “Выполнить”, выводится результат запроса. Также есть возможность увидеть сам запрос, нажав “Показать запрос” (см. Рисунок 9).

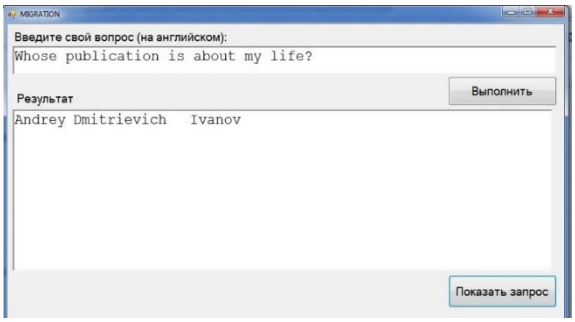


Рисунок 9. Интерфейс программы без показа запроса

По нажатии “Спрятать запрос” (см. Рисунок 10) можно убрать поле с выводом запроса.

Проект можно скачать по ссылке https://github.com/shparutask/TranslationSystem. Инструкция по запуску и пользованию

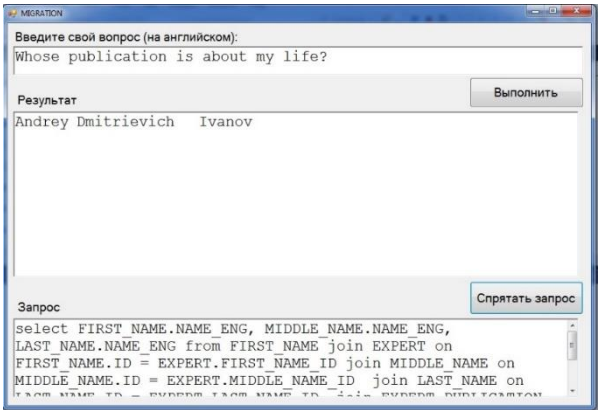


Рисунок 10. Интерфейс программы с показом запроса

приложением находится в Приложении 2.

ПРИМЕРЫ

Заключение

Итак, на данный момент есть приложение для Windows, настроенное на конкретную базу, которое выдает ответы на вопрос пользователя на русском и английском языках. Реализованы вопросы, указанные в приложении 1.

Используемая литература

1. Alexander Ran, Raimondas Lencevicius. Natural Language Query System for RDF Repositories // Proceedings of the 7-th International Symposium on Natural Language Processing, SLNP. –2007. –6.
2. Alessandra Giordani and Alessandro Moschitti. Generating SQL Queries Using Natural Language Syntactic Dependencies and Metadata // Department of Computer Science and Engineering University of Trento. –2012. – 6.
3. Florin Brad, Radu Iacob, Ionel Hosu, and Traian Rebedea. Dataset for a Neural Natural Language Interface for Databases (NNLIDB) // Proceedings of the 8-th International Joint Conference on Natural Language Processing. – 2017. – c.906-914.
4. Fred Popowich, Milan Mosny, David Lindberg. Interactive Natural Language Query Construction for Report Generation // Proceedings of the 7-th International Natural Language Generation Conference. –2012. – c.115-119.
5. Ikshu Blalla, Archit Gupta. Generating SQL queries from natural language // Department of Computer Science of Stanford University. – 2017. – 9.
6. Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. SQLizer: Query Synthesis from Natural Language // ACM Program. Lang. 1, 1, Article 1 (January 2017). – 2017. – 25.
7. Nicolas Kuchmann-Beauger. Question Answering System in a Business Intelligence Context // HAL archives-ouvertes.fr. –2017. – c.15-137.
8. Saima Noreen Khosa, Muhammad Rizwan. Database schema independent architecture for NL to SQL query conversion // Khwaja Fareed University of Engineering and IT. –2014. – c. 95-99.
9. Shadi Abdul Khalek, Sarfraz Khurshid. Automated SQL Query Generation for Systematic Testing of Database Engines // The University of Texas at Austin. –2010. – c.2-5.
10. Shay Cohen, Toms Bergmanis. A Natural Language Query System in Python/NLTK. –

https://github.com/andrrra/Natural-Language-Query-System.

11. Xiaojun Xu, Chang Liu, Dawn Song. SQLNet: Generating structured queries from Natural Language without reinforcement learning // ICLR-2018. –2017. – 15.

12. Посевкин Р. В. Модели, методы и программные средства построения естественно-языкового пользовательского интерфейса к базам данных // СПб-2018. -138