Р.А. СИМАКОВ, И.А. ЕРЕМИН

Муромский институт (филиал) ФГБОУ ВО «Владимирский государственный университет имени А.Г. и Н.Г. Столетовых», г. Муром

Разработка утилиты миграции БД с Oracle на Ред База Данных

УДК 004.021

В статье предлагается метод преобразования sql-скриптов создания базы данных с помощью дерева разбора, получаемого в результате синтаксического анализа скрипта.

Введение

Язык SQL стал стандартом для реляционных баз данных. Существует несколько спецификаций SQL, но ни одна из них не удовлетворяет всем требованиям разработчиков и пользователей к обработке данных. Если неукоснительно следовать стандартам, можно столкнуться с проблемой производительности, и придется обращаться к специальным функциям, обеспечиваемым поставщиком платформы. Побочный эффект – несовместимость решений, означающая для потребителей либо выбор в пользу конкретной платформы, либо программирование с ориентацией на несколько серверов баз данных. Специфичны для каждого из серверов баз данных логическая и физическая организация хранения данных, соответствующий набор операторов для управления сервером и объектами базы данных. Поэтому при миграции возникает множество вопросов, которые необходимо решить.

В настоящее время на государственном уровне отдается приоритет отечественному программному обеспечению, поэтому часто возникают задачи по переносу решений, использующих иностранные разработки. В связи с необходимостью миграции БД с Oracle на Ред База Данных, появилась идея создать утилиту,

которая позволит автоматизировать большинство операций преобразования.

В данной статье предлагается метод преобразования sql-скриптов с помощью дерева разбора, получаемого в результате синтаксического анализа скрипта.

1. Выбор подхода к решению задачи

Общий принцип работы утилиты предполагался следующий:

1. выполнение синтаксического анализа скрипта создания базы данных Oracle;
2. построение синтаксического дерева по полученным данным;
3. обход дерева с преобразованием отдельных узлов, в результате чего будет сформирован скрипт создания базы данных на РБД.

Процесс группировки символов исходного текста в слова или лексемы называется лексическим анализом, а программа, выполняющая его – лексером. Лексер группирует лексемы по типам (например, целые числа, идентификаторы, вещественные числа и т.д.). Каждая лексема описывается по крайней мере двумя свойствами: типом лексемы и соответствующим этому типу текстом из исходного файла [1].

Синтаксический анализатор (парсер) принимает на вход поток лексем и выполняет распознавание структуры выражений. В результате строится синтаксическое дерево (дерево разбора), которое наглядно показывает, каким образом выполнен разбор исходного текста. На рисунке 1 представлены основные этапы распознавания.

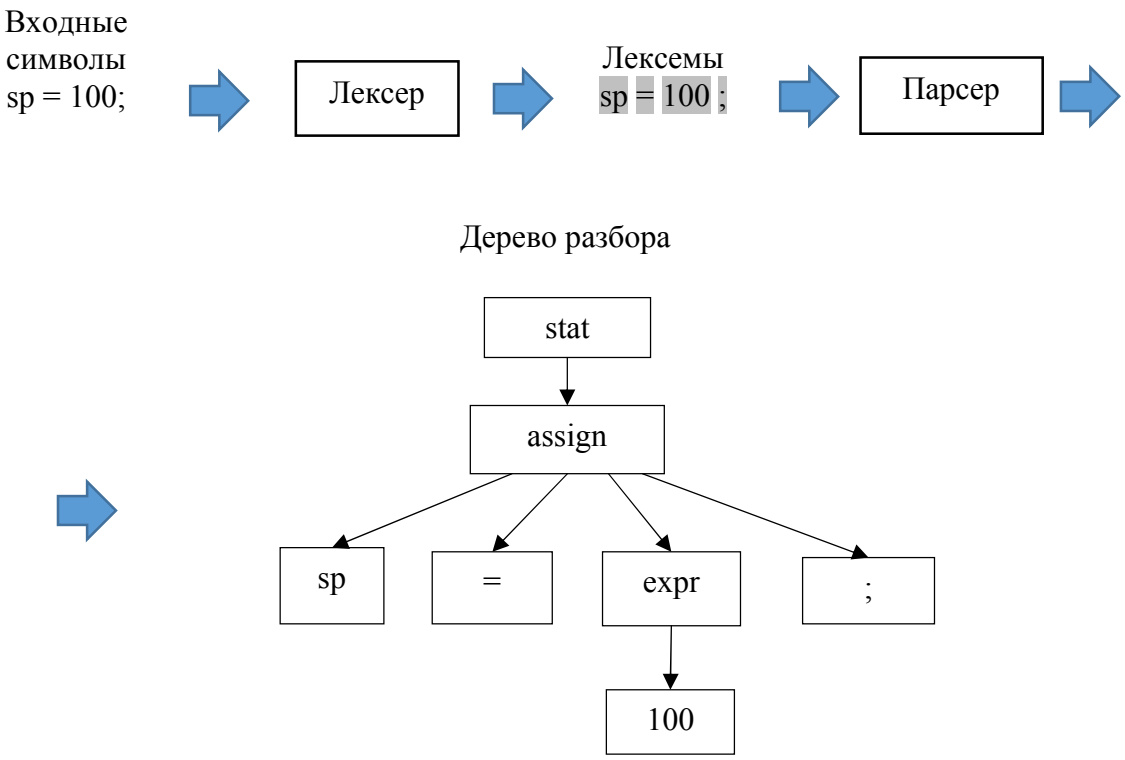

Рис. 1. Процесс разбора исходного текста.

Написание модулей, разбирающих эти данные, вручную является очень трудоемкой задачей и не всегда гарантирует хороший результат – результат, как правило, представляет собой большой по объему код, который достаточно сложно отлаживать. Существуют специальные средства, позволяющие облегчить создание парсеров – генераторы синтаксических анализаторов. Подобные средства оказываются незаменимыми, если необходимо использовать в работе программы данные какого-нибудь сложного формата [2].

Для создания синтаксического анализатора грамматики Oracle был выбран генератор анализаторов для формальных языков ANTLR4 [3]. Такой выбор обоснован тем, что в ANTLR4 есть возможность по описанной грамматике сгенерировать набор триггеров, каждый из которых соответствует какому-либо синтаксическому правилу. При обходе дерева разбора производится вызов того триггера, который соответствует текущему типу узла. Внутри триггера можно выделять из общего потока лексем только те, которые принадлежат текущему синтаксическому правилу. Над полученными лексемами можно выполнять операции вставки,

замены и удаления. Такой подход с триггерами показался наиболее логичным решением, т.к. грамматики Oracle и РБД по большей части схожи.

ANTLR генерирует код лексера и парсера, используя файл с описанием грамматики разбираемого языка, который состоит из лексических и синтаксических правил, описанных в расширенной форме Бэкуса – Наура [4]. Грамматика создается таким образом, что одни синтаксические правила определяются через другие. Лексер преобразует поток символов в поток лексем в соответствии с лексическими правилами, а парсер обрабатывает полученный поток лексем в соответствии с синтаксическими правилами.

Пример описания оператора «CREATE TABLE» в соответствии с грамматикой Oracle:

```
create_table
    : CREATE TABLE (schema_name PERIOD)? tableview_name
      '(' field_spec (',' field_spec)* ')'
      physical_properties?
      lob_storage_clause* ';' ;
```

Данное синтаксическое правило можно изобразить в виде синтаксической диаграммы (рисунок 2).

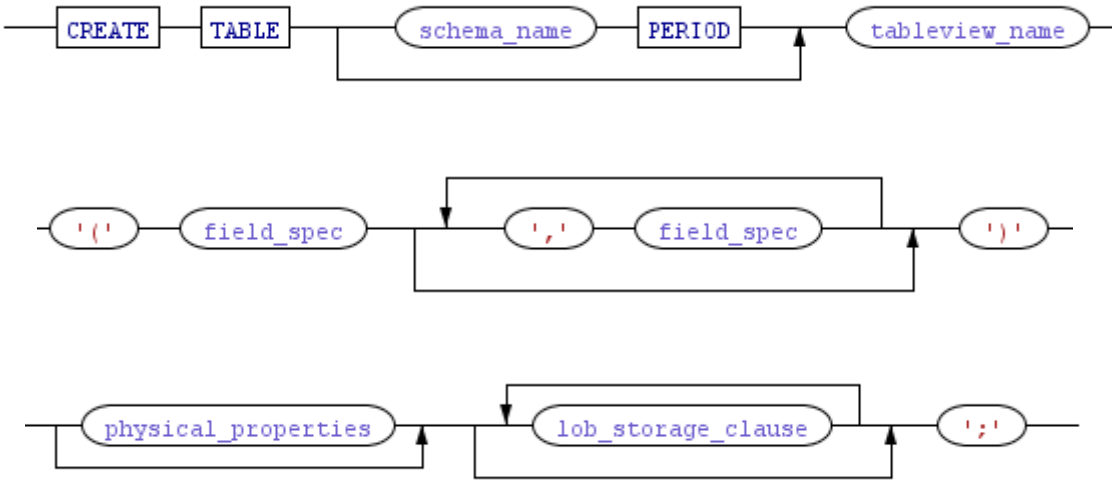

Рис. 2. Синтаксическая диаграмма оператора «CREATE TABLE».

ANTLR позволяет генерировать код на различных языках, среди которых Java и C#. Сгенерированные классы лексера и парсера подключаются в основную программу.

После реализации всех необходимых правил грамматики полученный файл можно использовать для генерации классов анализатора с помощью ANTLR4. В результате работы обычно генерируется следующий набор файлов (первая часть имени — это название грамматики):

– «plsqlLexer.java» – файл, содержащий определение класса лексического анализатора;

– «plsqlParser.java» – файл, содержащий определение класса синтаксического анализатора;

– «plsqlListener.java» – содержит интерфейс с объявлениями триггеров для всех синтаксических правил грамматики;

– «plsqlBaseListener.java» – содержит класс, реализующий интерфейс «plsqlListener», с пустыми определениями триггеров.

2. Алгоритм и правила преобразования

2.1. Первый обход дерева разбора

Некоторые отличия в грамматиках требуют предварительного сбора информации еще до начала выполнения преобразований. Для того, чтобы обойти дерево разбора, построенное для входного скрипта Oracle, и предварительно собрать нужную информацию, необходимо создать класс «InitialListener», который наследует «plsqlBaseListener», и в нем переопределить только те триггеры, которые относятся к синтаксическим правилам, в контексте которых должен выполняться сбор информации.

Рассмотрим конкретные случаи.

1. ALTER TABLE

В контексте данного оператора может изменяться состояние флага «NOT NULL» для столбца таблицы. В РБД такая операция не поддерживается, поэтому необходимо предварительно собрать информацию и уже на этапе генерации добавить «NOT NULL» для соответствующих столбцов в «CREATE TABLE».

Чтобы получить список столбцов, у которых должен быть установлен флаг «NOT NULL», для каждой таблицы, необходимо переопределить триггер «enterAlter_table». Внутри этого триггера

необходимо проверить наличие в контексте «ALTER TABLE» команды на установку «NOT NULL».

Контекст проверяется на наличие лексем «NOT» и «NULL». Если лексемы присутствуют, то извлекается название таблицы, к которой применяется «ALTER TABLE», и приводится к верхнему регистру, чтобы избежать сохранения одного и того же названия в разном регистре. Если название окружено двойными кавычками, то они убираются, так как не несут важной информации и хранить их не требуется. Таким же образом извлекается название изменяемого столбца. Затем полученное название добавляется в список для текущей таблицы или создается новый список, если команды модификации столбцов этой таблицы еще не встречались. Полученный ассоциативный массив «table_map», который хранит соответствия «Таблица – Список столбцов», используется в дальнейшем во время генерации выходного скрипта для добавления внутри оператора «CREATE TABLE» флагов «NOT NULL» для перечисленных в массиве столбцов.

Также в контексте «ALTER TABLE» необходимо учесть создание первичных ключей. Если для столбца (столбцов) создается первичный ключ, то для него (них) тоже необходимо установить флаг «NOT NULL».

В контексте правила выполняется поиск лексемы «PRIMARY». Если лексема присутствует, то извлекаются название таблицы и названия столбцов рассмотренным выше способом. Эта информация также сохраняется в тот же ассоциативный массив «table_map».

2. CREATE INDEX

Обычно из БД Oracle экспортируются все индексы, даже те, которые были созданы автоматически вместе с ключами и ограничениями уникальности. Во время преобразования необходимо их убрать, чтобы избежать повторного создания одного и того же индекса. Для этого предварительно собирается информация по ключам и ограничениям уникальности, присутствующим в исходном скрипте.

В контексте правила «ALTER TABLE» выполняется поиск лексем «PRIMARY», «FOREIGN» и «UNIQUE». Если одна из этих

лексем присутствует, то это значит, что в результате выполнения текущего оператора автоматически будет создан индекс. В этом случае извлекается название ограничения, которое соответствует названию автоматически созданного индекса. Это название добавляется в список «index_names». Все индексы, попавшие в этот список, в дальнейшем будут удалены из исходного скрипта.

2.2. Второй обход дерева разбора

Для второго обхода дерева разбора, в результате которого уже будут вноситься изменения в исходный скрипт, необходимо создать класс «RewritingListener», который наследует «plsqlBaseListener», и в нем переопределить только те триггеры, которые относятся к синтаксическим правилам, требующим преобразования.

1. Схема БД Oracle

В одной БД Oracle есть несколько схем. Например, системные таблицы располагаются в одной схеме, а пользовательские – в других. Схема – это своего рода контейнер хранимых в БД объектов одного пользователя. Обычно при экспорте извлекаются метаданные только тех объектов, владельцем которых является пользователь, под которым выполнено подключение. То есть извлекается конкретная схема БД. В полученном скрипте перед именами объектов указывается имя этой схемы [5]. В РБД понятие схемы не используется, поэтому при преобразовании ее имя убирается [6]. При необходимости можно рассмотреть случай, когда в одном скрипте экспортированы несколько схем БД. Но нужно учесть, что в разных схемах БД могут быть таблицы с одинаковыми названиями. То же касается и других объектов БД.

2. Преобразование типов данных

Для типа данных «NUMBER» явно устанавливается максимальная точность «18», если вместо нее указана «*». Если для типов «NUMBER» и «NUMERIC» точность и масштаб не указаны, то для них явно устанавливаются значения по умолчанию «(18, 4)». Если для типа «FLOAT» указана точность, то он заменяется на «DOUBLE PRECISION». Если для типа «TIMESTAMP» указана точность, то она удаляется. Если для типов

«VARCHAR» и «VARCHAR2» не указан размер, то устанавливается размер по умолчанию «(250)».

Преобразования типов данных сведены в таблицу 1.

Таблица 1

Преобразования типов данных

Oracle	Ред База Данных
NUMBER(n,m)	NUMERIC(n,m)
VARCHAR2(n BYTE)	VARCHAR(n)
NVARCHAR2(n)	VARCHAR(n)
NCHAR(n)	CHAR(n)
NUMBER(*,0)	NUMERIC(18,0)
FLOAT(n)	DOUBLE PRECISION
BINARY_FLOAT	FLOAT
BINARY_DOUBLE	DOUBLE PRECISION
TIMESTAMP(n)	TIMESTAMP
CLOB	BLOB SUB_TYPE 1
RAW	BLOB

3. CREATE TABLE

Удаляются физические параметры хранения таблиц, устанавливаются флаги «NOT NULL» для столбцов из списка, полученного во время первого обхода дерева разбора.

4. CREATE INDEX

Оператор «CREATE INDEX» удаляется, если название создаваемого индекса находится в списке автоматически созданных индексов. В противном случае удаляются физические параметры индекса.

В Oracle можно создавать индексы с использованием выражений вместо столбцов, причем таких выражений может быть несколько. Например:

```
CREATE INDEX upper_ix ON employees (UPPER(last_name),
UPPER(first_name));
```

В РБД индексирование нескольких выражений не поддерживается. Нужно определить, имеет ли смысл при преобразовании разбивать индекс на несколько в подобных случаях.

5. Генераторы (последовательности)

В контексте оператора «CREATE SEQUENCE» удаляются неподдерживаемые РБД параметры генератора. Вместо параметра «START WITH», устанавливающего начальное значение генератора, добавляется отдельная команда «SET GENERATOR».

6. CREATE VIEW

В контексте оператора «CREATE VIEW» производится замена «REPLACE» на «ALTER», удаляется неподдерживаемый РБД параметр «FORCE». В Oracle данный параметр позволяет создать представление, даже если оно ссылается на несуществующие таблицы или представления. В связи с этим, необходимо дополнительно реализовать упорядочивание операторов «CREATE VIEW», чтобы не возникало ошибок при их выполнении.

7. Контекстные переменные

Заменяются контекстные переменные «SYSTIMESTAMP» на «CURRENT_TIMESTAMP» и «SYSDATE» на «CURRENT_DATE».

8. Встроенные функции

```
REPLACE(строка_символов, строка_поиска, [стро-
ка_замены])
```

Функция «REPLACE» выполняет замену подстроки на другую подстроку в указанной строке. В Oracle параметр, содержащий подстроку замены, может отсутствовать (подразумевается пустая подстрока). В этом случае искомая подстрока удаляется. В РБД необходимо явно указывать пустую строку в подобной ситуации. Поэтому если в исходном скрипте функция вызывается с 2-мя параметрами, то при преобразовании добавляется 3-й.

Функция «LENGTH» в Oracle возвращает длину указанной строки. В РБД подобная функция имеет имя «CHAR_LENGTH». Поэтому при преобразовании выполняется замена имени функции.

9. CREATE PROCEDURE

На данном этапе преобразование языка процедур не реализовано, но вместо этого создаются так называемые «заглушки», где тело процедуры заключено в комментарий. Перед оператором добавляется команда установки терминального символа «SET TERM ^ ;», а после – обратная команда «SET TERM ; ^». Это необходимо из-за того, что в теле процедуры инструкции завершаются точкой с запятой, а сам оператор «CREATE PROCEDURE» должен завершаться символом, отличным от точки с запятой.

10. CREATE FUNCTION

Функция отличается от процедуры тем, что она предназначена для возврата значения, которое может использоваться в более крупном операторе. В РБД версии 2.6 функции отсутствуют (появились в 3.0), но вместо этого процедуры могут работать так же, как и функции. Поэтому при преобразовании «CREATE FUNCTION» заменяется на «CREATE PROCEDURE».

11. CREATE TRIGGER

Преобразование реализовано аналогично «CREATE PROCEDURE».

12. Комментарии

На данный момент, комментарии остаются после преобразования. Возможно, лучшим решением будет удаление комментариев из скрипта, поскольку часто встречается случай, когда часть скрипта, к которому относится комментарий, удаляется. В утилиту можно добавить опцию, с помощью которой будет возможность оставить комментарии.

3. Пример работы утилиты

Этот несложный пример скрипта Oracle позволит показать, какие преобразования выполняет программа.

```
CREATE TABLE "SCHEMA"."TABLE"
(
    "ID"            NUMBER(15,0),
    "KEY"           VARCHAR2(100),
```

```
    "ACTUAL"        NUMBER(1,0) DEFAULT 1,
    "TYPE"          NUMBER(*,0),
    "FACTOR"        FLOAT(126),
    "DATE"          TIMESTAMP(6),
    "DESCRIPTION"   CLOB
);

ALTER TABLE "SCHEMA"."TABLE" ADD CONSTRAINT "PK_TABLE"
PRIMARY KEY ("ID");
CREATE UNIQUE INDEX "SCHEMA"."PK_TABLE" ON "SCHE-
MA"."TABLE" ("ID");
CREATE SEQUENCE  "SCHEMA"."TABLE_SEQ" START WITH 281;
```

Утилита упакована в JAR-файл. При запуске необходимо передать ей в качестве аргумента путь к файлу с тестовым скриптом, например, «test.sql». На выходе получаем преобразованный скрипт, который можно выполнять на РБД (рисунок 3).

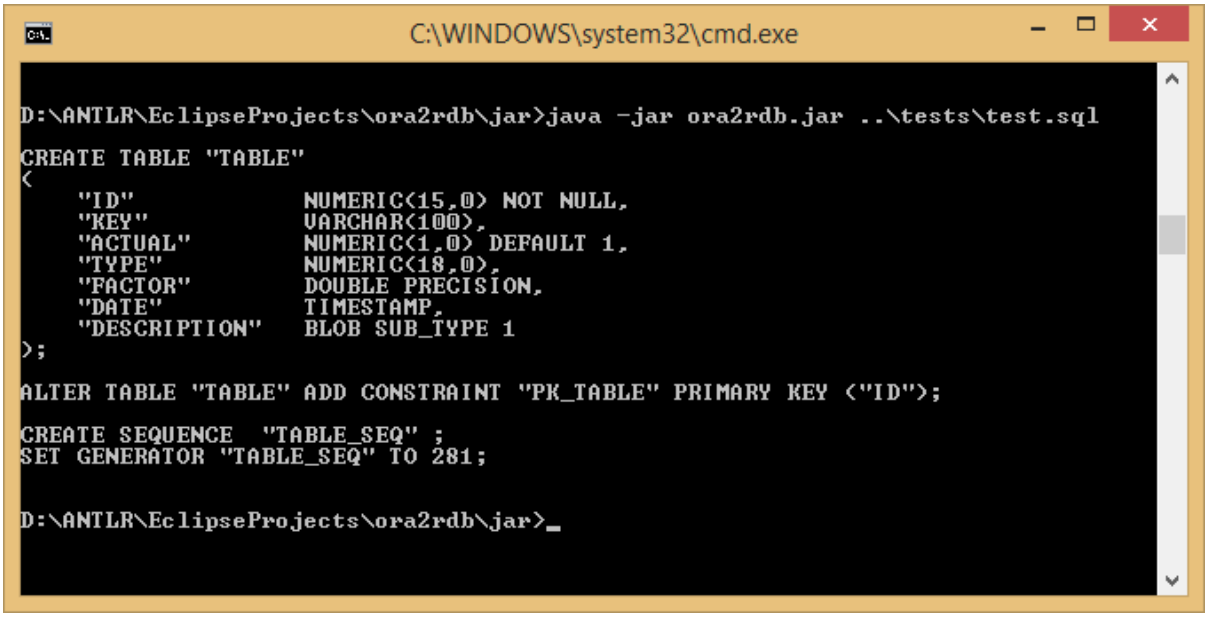

Рис. 3. Результат работы утилиты.

В таблице 2 приведено сравнение исходного и преобразованного скриптов. Части скрипта, измененные по описанным выше правилам, выделены курсивом.

Таблица 2

Сравнение исходного и преобразованного скриптов

Oracle	Ред База Данных

CREATE TABLE *"SCHEMA".*"TABLE"
(
  "ID" ***NUMBER***(15,0),
  "KEY" VARCHAR***2***(100),
  "ACTUAL" ***NUMBER***(1,0) DEFAULT 1,
  "TYPE" ***NUMBER***(******,0),
  "FACTOR" ***FLOAT(126)***,
  "DATE" TIMESTAMP***(6)***,
  "DESCRIPTION" ***CLOB***
);
 	
ALTER TABLE *"SCHEMA".*"TABLE"
ADD CONSTRAINT "PK_TABLE" PRIMARY KEY
("ID");
 	
***CREATE UNIQUE INDEX "SCHE-
MA"."PK_TABLE"
ON "SCHEMA"."TABLE" ("ID");***
 	
CREATE SEQUENCE *"SCHEMA".*"TABLE_SEQ"
START WITH 281;
	CREATE TABLE "TABLE"
	(
	  "ID" ***NUMERIC***(15,0) ***NOT NULL***,
	  "KEY" VARCHAR(100),
	  "ACTUAL" ***NUMERIC***(1,0) DEFAULT 1,
	  "TYPE" ***NUMERIC***(***18***,0),
	  "FACTOR" ***DOUBLE PRECISION***,
	  "DATE" TIMESTAMP,
	  "DESCRIPTION" ***BLOB SUB_TYPE 1***
	);
	
	ALTER TABLE "TABLE"
	ADD CONSTRAINT "PK_TABLE" PRIMARY KEY
	("ID");
	
	CREATE SEQUENCE "TABLE_SEQ" ;
	
	SET GENERATOR "TABLE_SEQ" TO 281;

4. Проблемы подхода с триггерами

В процессе написания программы с использованием подхода с триггерами возникли некоторые проблемы:

1. Нет контроля над форматированием выходного скрипта. Части исходного скрипта, не требующие преобразования, отправятся в выходной скрипт без изменения форматирования. Если форматирование было неоднородным, то после преобразования оно таким и останется.

2. После удаления неиспользуемых лексем в скрипте остаются пробелы, переходы на новую строку и т.п., которые были до и после них.

3. Невозможно контролировать порядок вставки новых лексем. Например, если синтаксическое правило состоит из двух вложенных правил, то позиция вставки в конец первого правила соответствует позиции вставки в начало второго правила. Если в триггерах для этих правил выполняются вставки в эту позицию, то итоговый порядок вставленных лексем будет зависеть от порядка вызова триггеров, без учета принадлежности каждой лексемы к какому-либо правилу. Проблема осложняется, если вставки в эту позицию

производятся из триггеров правил, находящихся на разных уровнях вложенности.

4. Отсутствует возможность переупорядочивания операторов. Реализация этой возможности необходима, например, для решения описанной выше проблемы с упорядочиванием «CREATE VIEW».

Литература

1. Terence Parr. The Definitive ANTLR Reference, The Pragmatic Bookshelf, 2012. –322 с.

2. Написание парсеров с помощью ANTLR // URL: http://club.shelek.ru/viewart.php?id=39 (дата обращения: 31.03.16)

3. ANTLR // URL: http://www.antlr.org/ (дата обращения: 31.03.16)

4. Форма Бэкуса – Наура // URL: https://ru.wikipedia.org/wiki/Форма_Бэкуса_—_Наура (дата обращения: 05.04.16)

5. Oracle Database Online Documentation 11g Release 2 (11.2) // URL: http://docs.oracle.com/cd/E11882_01/index.htm (дата обращения: 11.04.16)

6. Ред База Данных 2.6. Руководство по SQL, 2015. –388 с.