**BC/NW 2021№ 2 (38):12.1**

**РАЗРАБОТКА УТИЛИТЫ ДЛЯ ПРЕОБРАЗОВАНИЯ SQL ЗАПРОСОВ И ВИЗУАЛИЗАЦИИ ИСХОДНОЙ СХЕМЫ БАЗЫ ДАННЫХ**

Абросимов Л.И., Пичугин Д.А

С каждым годом увеличивается объем генерируемой человеком информации. К 2020 году объем хранимых данных увеличился примерно до 40 зеттабайт (1 ЗБ ~ 1 миллиард ГБ). К 2025 году прогнозируется увеличение до 400 зеттабайт. Соответственно, управление структурированными и неструктурированными данными при помощи современных технологий — сфера, которая становится все более важной. Интересуются большими данными как отдельные компании, так и целые государства.

Аналитика данных позволяет выявлять крайне ценную информацию из наборов данных. Благодаря этому бизнес, например, может определять тенденции, прогнозировать производственные показатели и оптимизировать собственные расходы.

Однако перед тем, как данные станут пригодны для анализа, они должны пройти процесс подготовки. Под подготовкой данных подразумевается их сбор, структурирование, очистка от избыточной информации, выгрузка в аналитические системы и многое другое.

При формировании новых наборов данных, либо оптимизации существующих возникает задача анализа взаимосвязей объектов базы данных или хранилища. В случае, когда нет визуального представления этой базы данных, а информацию о таблицах и их отношениях можно получить лишь из SQL запросов, задача анализа сильно усложняется. В таком случае возникает потребность в инструменте, помогающем получить наглядное представление базы данных.

В рамках данного проекта осуществлена разработка программы, генерирующей графическую модель из SQL DDL скриптов, и ее интерфейса.

**Анализ предметной области и постановка задачи.**

***Предметная область***

Компания «Кадровый учет» входит в концерн «Объединение» и занимается кадровым администрированием всего концерна. Для хранения информации о сотрудниках применяется СУБД Oracle Database. Одной из задач компании является подготовка срезов данных по базе данных сотрудников для их предоставления в подразделения концерна, занимающиеся бизнес-аналитикой.

При проектировании срезов данных, либо при оптимизации существующей базы данных возникает перед аналитиками и разработчиками компании встает задача анализа текущей организации данных.

Анализ структуры базы данных – довольно трудоемкий процесс. Для упрощения этой задачи, данные необходимо визуализировать. Визуализация данных — это представление данных в виде, который обеспечивает наиболее эффективную работу человека по их изучению. Наиболее распространенный и информативный вид визуализации данных – это графики и диаграммы.

В данном случае, необходим инструмент, позволяющий получить графическую модель базы данных, используя SQL скрипты для создания таблиц, а также их связей (DDL).

***Анализ предметной области***

Визуальное представление базы данных должно быть достаточно информативным и не содержать избыточной информации. Оптимальный вид графической модели – при котором она соответствует этим критериям, легка для восприятия и анализа (рис. 1.).

Графическая нотация схемы представлена на рис. 1:

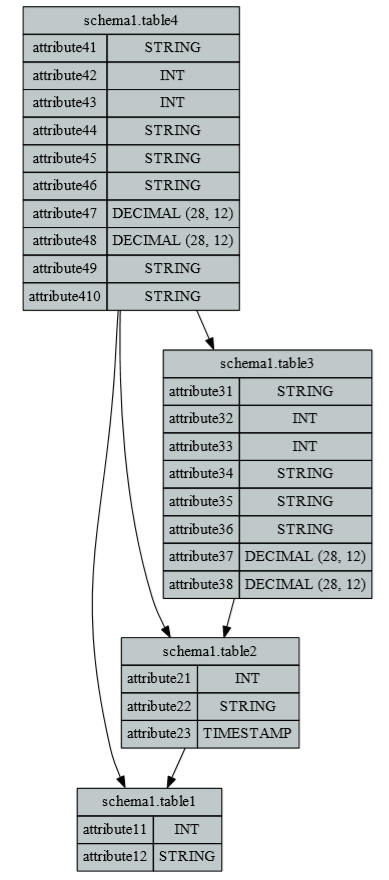


Рис. 1 Пример графической модели базы данных

* Совокупность таблиц и их отношения представлена в виде ориентированного графа.
* В вершинах графа находятся таблицы, ребрами обозначаются зависимости таблиц.
* В каждой вершине указано название схемы, в которой содержится таблица, название самой таблицы, названия и типы столбцов.
* Направление рёбер обозначает следующее: таблица, лежащая в вершине, от которой исходит ребро, зависит от таблицы, лежащей в вершине, в которую оно приходит. Например, таблица table1 зависит от таблицы table2, таблицы table1, table2, table3 зависят от таблицы table 4.

Стоит отметить, что такая схема напоминает визуальное представление базы данных иерархического типа [2]. Однако в таком виде может быть представлена любая база данных.

***Анализ существующих решений***

Из множества инструментов для работы с Oracle DB в первую очередь стоит отметить Oracle SQL Developer [3]. Это интегрированная среда разработки на языках SQL и PL/SQL, с возможностью администрирования баз данных, ориентирована на применение в среде Oracle Database. Продукт предоставляется корпорацией Oracle бесплатно.

Oracle SQL Developer включает в себя инструмент для моделирования базы данных, позволяющий задавать таблицы с помощью SQL DDL скриптов. Однако, визуальные модели, полученные при помощи данного инструмента, имеют ряд недостатков. Модель, построенная этим инструментом приведена на рис. 2.

Из рис. 2 мы видим, что на визуальной схеме присутствуют пересечения стрелок, отображающих связи таблиц. Также видно, что скрипты типа CREATE TABLE AS SELECT не поддерживаются. Таблица, для построения которой использован скрипт данного типа находится в правом верхнем углу рисунка. Названия столбцов отображены не корректно, типы данных не распознаны.

Когда речь идет о выборке данных в виде отдельной таблицы, построение таблиц по типу CREATE TABLE AS SELECT является наиболее удобным способом. Поэтому поддержка данного типа скриптов – важный критерий при выборе инструмента для моделирования базы данных.

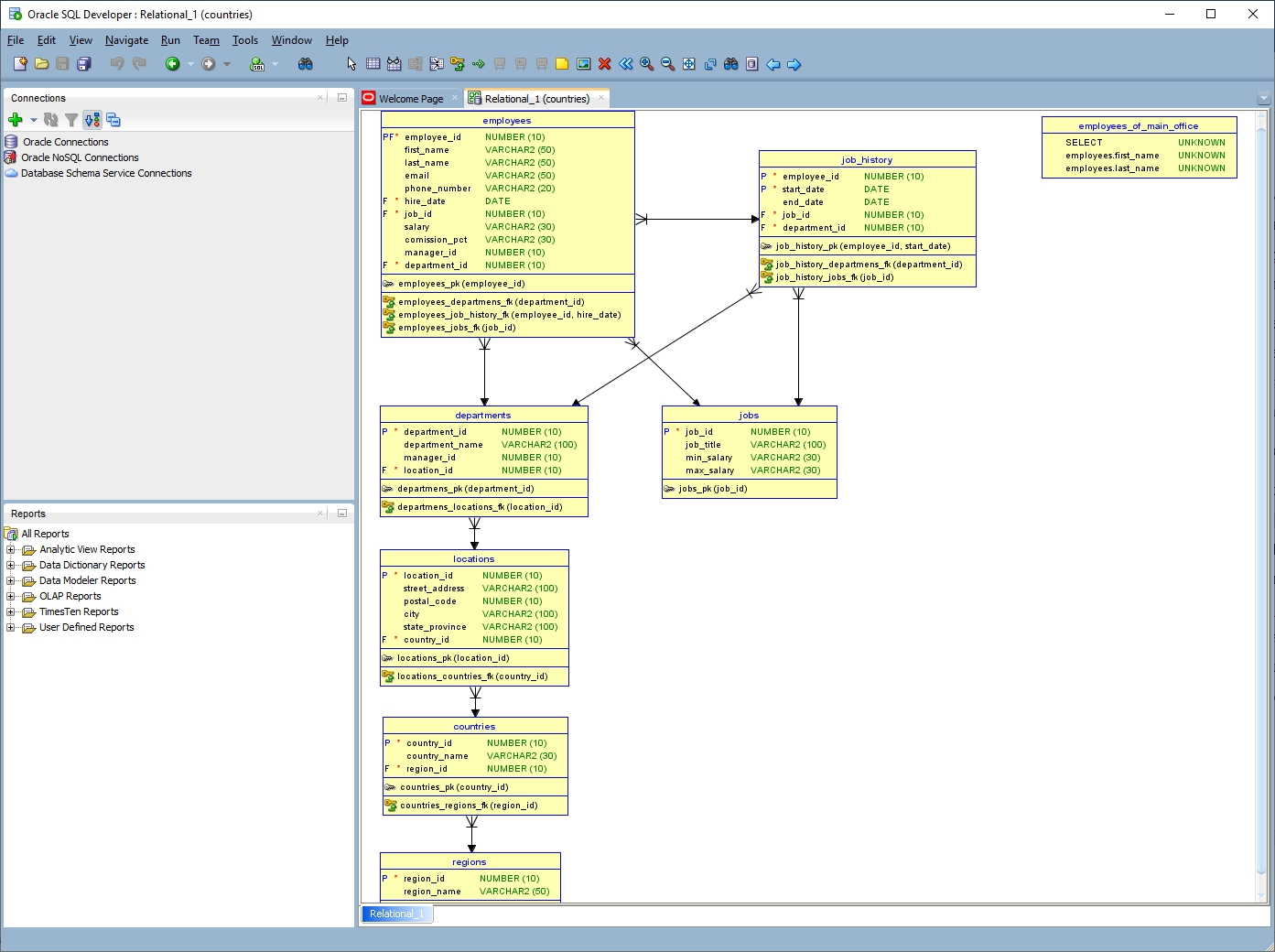


Рис. 2 Пример модели, построенной в Oracle SQL Developer

Также для решения подобных задач существует множество аналогичных инструментов для разработки и администрирования баз данных [4]. Наиболее популярные из них это Dataedo, SchemaSpy, DataGrip, Navicat. Данные продукты позволяют построить диаграмму базы данных, основываясь на её метаданных.

Dataedo, SchemaSpy и DataGrip имеют существенный недостаток: в них нет возможности добавления на визуальную модель базы данных еще не существующих в базе данных таблиц [5-7]. При отсутствии у разработчика доступа к метаданным, данные пакеты не подойдут для решения задачи визуализации.

Построение визуальных моделей из SQL запросов реализовано в Navicat [8].

Среди плюсов Navicat стоит также выделить возможность одновременного использования до 7 различных источников данных [9]. Он поддерживает наиболее популярные СУБД: MySQL, MariaDB, MongoDB, SQL Server, Oracle, PostgreSQL, и SQLite. Также данный пакет совместим с облачными хранилищами, такими как Amazon RDS, Amazon Aurora, Amazon Redshift, Microsoft Azure, Oracle Cloud, Google Cloud и MongoDB Atlas [8]. Однако так как данный продукт помимо построения диаграммы имеет множество других возможностей, приобретение его лишь для визуализации для некоторых компаний может быть нерациональным решением в силу стоимости Navicat, которая на данный момент составляет свыше 1200 долларов для полной коммерческой лицензии данного продукта.

***Постановка задачи***

Основными задачами являются:

* разработка алгоритма и программы для построения визуальной схемы базы данных из SQL запросов.
* разработка удобного интерфейса для управления программой.

Разрабатываемое приложение должно удовлетворять следующим требованиям:

* должен быть предусмотрен пользовательский интерфейс, позволяющий ввести исходные данные;
* программа должна получать на вход DDL (Data Definition Language) SQL запросы в виде текстовых файлов;
* программа должна предоставлять пользователю графическую модель на выходе;
* графическая модель должна быть информативной и не содержать избыточной информации;
* входные запросы должны полностью соответствовать синтаксису PL/SQL;
* программа должна быть написана на языке Java.

**Разработка алгоритма и программы, реализующей приложение**

***Выбор средств разработки и обоснование алгоритма***

Для построения визуальной модели используется графический пакет GraphViz [9]. Выбор пакета обусловлен его широким функционалом, простотой в использовании, доступностью и наличием подробной документации. Данный графический пакет предназначен для визуализации графов. GraphViz равномерно распределяет вершины и ребра графа, а также сводит к минимуму количество пересечений ребер, что позволяет получить легко читаемую визуализацию.

В GraphViz используется язык описания графов DOT. На рис. 3 приведен пример графа, описанного с помощью DOT.

***graph graphname {***

***// label - видимое название вершины***

***a [label="Foo"];***

***// shape - определение формы вершины***

***b [shape=box];***

***// color - определение цвета ребра***

***a -- b -- c [color=blue];***

***// style - определение стиля ребра***

***b -- d [style=dotted];***

***}***

Рис. 3 Пример графа, описанного с помощью DOT

На рис. 4 представлен результат обработки данной конструкции пакетом GraphViz.

Для представления базы данных в виде графа на языке DOT, информацию о таблицах из SQL DDL запросов необходимо извлечь и структурировать.

Преобразование SQL запросов происходит в два этапа. Сначала проводится синтаксический анализ и представление запросов как дерева объектов. Затем на его основе происходит визуализация и представление в виде графической модели.

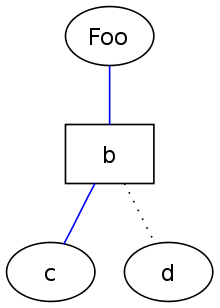


Рис. 4 Пример графа, построенного в GraphViz

Согласно техническому заданию, программа должна быть разработана на языке высокого уровня Java на платформе Java SE. В качестве редактора кода выбрана среда разработки IntelliJ IDEA [10]. В данной среде разработки обеспечена полная поддержка Java SE, а также интеграция с наиболее популярными системами контроля версий, что облегчает разработку и отладку приложения. Для автоматизации сборки проекта используется фреймворк Maven.

Большая часть необходимого функционала для синтаксического анализа реализована в фреймворке JSqlParser. Данный фреймворк позволяет в результате синтаксического анализа скриптов SQL, представить их в виде дерева объектов (частей запроса, таблиц, столбцов и т.д.) [11]. Однако данный фреймворк не поддерживает запросы типа CREATE AS SELECT, а также не распознает зависимости таблиц. За основу синтаксического анализа в программе взят данный фреймворк, а также реализован весь недостающий функционал.

Фреймворк JSqlParser поддерживает следующий синтаксис для запросов CREATE TABLE (рис. 5).

***CREATE TABLE table\_name***

***(***

***column1 datatype [ property1, property2, … propertyn ],***

***column2 datatype [ property1, property2, … propertyn ],***

***…***

***column\_n datatype [ property1, property2, … propertyn ],***

***);***

Рис. 5 Синтаксис запросов CREATE TABLE, поддерживаемых JSqlParser

Под [ property1, property2, … propertyn ] подразумевается список свойств столбца.

Одним из существенных недостатков данного ограничения синтаксиса – неспособность распознавания внешних ключей таблицы (при добавлении в запрос строки ***CONSTRAINT constraint\_name FOREIGN KEY(column1) REFERENCES table\_name(column2)***, JSqlParser решит, что это столбец с названием ***CONSTRAINT***, типов данных ***constraint\_name*** и свойствами ***FOREIGN KEY(column1) REFERENCES table\_name(column2)***). Поскольку для данного типа запросов связи таблиц определяются по внешним ключам, в программе должна присутствовать поддержка внешних ключей. Данный функционал доработан в программе, добавлена поддержка *CONSTRAINT* ограничений, в частности *FOREIGN KEY*, *PRIMARY KEY*. Поддерживаемый программой синтаксис CREATE TABLE запросов:

***CREATE TABLE table\_name***

***(***

***column1 datatype [ property1, property2, … propertyn ],***

***column2 datatype [ property1, property2, … propertyn ],***

***…***

***column\_n datatype [ property1, property2, … propertyn ],***

***[CONSTRAINT constraint\_name FOREIGN KEY(column1) REFERENCES table\_name(column2)***

***| CONSTRAINT constraint\_name PRIMARY KEY (column1, column2, … column\_n)]***

***);***

Рис. 6 Поддерживаемый программой синтаксис CREATE TABLE

Как сказано ранее, JSqlParser не поддерживает запросы типа CREATE TABLE AS SELECT. Однако фреймворк распознает стандартные SQL DML запросы типа SELECT. За основу преобразования CREATE TABLE AS SELECT запросов в программе взят функционал синтаксического анализа SELECT запросов с помощью JsqlParser.

Поддерживаемый программой синтаксис CREATE TABLE AS SELECT запросов:

***CREATE TABLE new\_table AS***

***(SELECT old\_table\_1.column\_1 as new\_column\_1,***

***old\_table\_2.column\_2 as new\_column\_2,***

***…***

***old\_table\_n.column\_n as new\_column\_n,***

***FROM old\_table\_1***

***[ LEFT | RIGHT | INNER | FULL ] JOIN old\_table\_2 ON {JOIN\_CLAUSE}***

***…***

***[ LEFT | RIGHT | INNER | FULL ] JOIN old\_table\_n ON {JOIN\_CLAUSE}***

***WHERE {WHERE\_CLAUSE}***

***ORDER BY {ORDER\_BY\_CLAUSE}***

***);***

Рис. 7 Поддерживаемый программой синтаксис CREATE TABLE AS SELECT

В CTAS запросах не поддерживаются агрегатные функции, а также предложения GROUP BY и HAVING.

При разработке использованы основные концепции объектно-ориентированного программирования [10].

***Основные классы и их назначение***

**SimpleGui –** реализует графический интерфейс пользователя.

**DiagramBuilder –** инициализирует модуль логирования, запускает синтаксический анализ и построение графической схемы.

**Parser** – выполняет синтаксический анализ, преобразует входные запросы в объекты.

**Session** – хранит массив таблиц для текущего процесса синтаксического анализа.

**MyTable** – экземпляры класса – преобразованные таблицы.

**Formatter** – отвечает за построение визуальной модели.

Диаграмма классов представлена на рис. 8. Она демонстрирует общую структуру иерархии классов, их столбцов, методов и взаимодействий между классами. Классы программы объединены в общий пакет vizualize. Класс SimpleGui находится во вложенном пакете visualize.gui, а классы Parser, DiagramBuilder, Session, MyTable, Formatter находятся в visualize.builder. Разделение классов по пакетам обеспечивает организацию классов в пространстве имен [12]. Классы пакета visualize.builder отвечают за программно-аппаратную часть приложения, а классы содержащиеся в visualize.gui отвечают за клиентскую сторону пользовательского интерфейса приложения.

***Описание полей и методов классов***

*SimpleGui*

Данный класс наследуется от JFrame. Класс JFrame включен в библиотеку Swing. Swing является частью Java Development Kit, реализует набор графических компонентов и предназначена для создания пользовательских интерфейсов. Класс JFrame реализует функционал построения окна с рамкой.



Рис. 8 Диаграмма классов программы

Модификатор доступа класса - public.

Таблица 1 – Поля класса SimpleGui

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Модификатор доступа | Тип | Описание |
| mainPanel | private | JPannel | Главное окно интерфейса |
| addFiles | private | JButton | Кнопка добавления файлов |
| runner | private | JButton | Кнопка запуска основной программы |
| textArea | private | JTextArea | Текстовое поле, в котором отображаются пути к входным файлам |
| fileChooser | private | JFileChooser | Диалоговое окно выбора файлов |
| paths | private | ArrayList<String> | Массив, хранящий пути к входных данным |

Таблица 2 – Методы класса SimpleGui

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Модификатор доступа | Входные параметры | Тип | Описание |
| SimpleGui | private | String title | - | Конструктор. Обработка действий пользователя |
| main | public | - | void | Начальная точка программы |
| $$$setupUI$$$ | private | - | void | Конфигурация свойств компонентов интерфейса |
| $$$getRootComponent$$$ | private | - | void | Системный метод |

Методы $$$setupUI$$$ и $$$getRootComponent$$$ сгенерированы автоматически редактором пользовательского интерфейса Intellij IDEA GUI Designer и являются обязательными для корректной работы интерфейса. Данный редактор является одной из базовых компонент используемой среды разработки.

*DiagramBuilder*

Модификатор доступа класса – public.

Таблица 3 – Поля класса DiagramBuilder

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Модификатор доступа | Тип | Описание |
| logger | private | Logger | Модуль логирования программы |

Таблица 4 – Методы класса DiagramBuilder

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Модификатор доступа | Входные параметры | Тип | Описание |
| DiagramBuilder | private | - | - | Конструктор. Инициализирует модуль логирования. |
| build | public | String[] args | File | Запускает синтаксический анализ и построение схемы. Возвращает файл с графической схемой |

*Parser*

Метод использует классы фреймворка JSqlParser. JSqlParser предоставляет базовый функционал синтаксического анализа SQL запросов с возможностью преобразования в запросов в дерево объектов. Наиболее используемые в программе классы JSqlParser приведенны в таблице 5.

Таблица 5 – Классы JSqlParser

|  |  |  |
| --- | --- | --- |
| Пакет | Класс | Описание |
| net.sf.jsqlparser.parser | CCJSqlParserManager | Использует JavaCC, проверяет текст и создает объекты. |
| net.sf.jsqlparser.schema | Table | Таблица. Содержит имя таблицы и схемы. |
| net.sf.jsqlparser.statement | Statement | SQL запрос |
| net.sf.jsqlparser.statement.create.table | CreateTable | SQL запрос Create Table |
| net.sf.jsqlparser.statement.create.table | ColumnDefinition | Столбец таблицы из SQL запроса. Содержит название, тип и свойства |
| net.sf.jsqlparser.statement.select | Select | SQL запрос Select |
| net.sf.jsqlparser.statement.select | PlainSelect | SQL Select, разбитый по группам операций |

Модификатор доступа класса – package private.

Таблица 6 – Поля класса Parser

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Модификатор доступа | Тип | Описание |
| logger | private | Logger | Модуль логирования программы |
| session | private | Session | Объект типа session для текущего запуска |
| createAsSelectPaths | private | ArrayList<String> | Массив путей к CTAS скриптам |

Таблица 7 – Методы класса Parser

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Модификатор доступа | Входные параметры | Тип | Описание |
| Parser | package-private | - | - | Конструктор. Инициализирует модуль логирования |
| parse | package-private | String[] args | void | Преобразует входной массив в объекты Statement |
| parse | private | Statement statement | void | Преобразует объект Statement в MyTable, добавляет его в session |
| checkExistence | private | Table table | boolean | Проверяет существование таблицы в текущем session |
| checkCAS | private | String path | boolean | Проверяет, является ли скрипт SQL CTAS |
| parseCAS | private | - | void | Преобразует SQL CTAS скрипты в объекты MyTable, добавляет в session |
| parseColumns | private | List<String> relations, List selectItems | ArrayList< ColumnDefinition> | Анализирует SQL скрипт, возвращает столбцы таблицы |
| getColumnDatatype | private | List<String> relations, String column | ColDataType | Выполняет поиск столбца в проанализированных таблицах, возвращает его тип. |
| findTable | private | List<String> relations, String colTableName | MyTable | Ищет таблицу из relations, имеющую столбец colTableName |
| findTable | private | String tableName | MyTable | Ищет таблицу в session по названию |
| findCol | private | MyTable table, String searchedColumn | ColDataType | Ищет столбец в таблице по названию, возвращает его тип |
| checkForeignKeys | private | CreateTable createTable | List<String> | Проверяет наличие внешних ключей в скрипте, возвращает список зависимостей |
| checkForeignKey | private | Object col | String | Проверяет наличие внешних ключей в скрипте. Возвращает одну зависимость |
| getRelatedTable | private | String columnDefinition | String | Извлекает из строки название зависимой таблицы |
| getSession | package-private | - | Session | Getter для session. |

*Session*

Модификатор доступа класса – package private.

Таблица 8 – Поля класса Session

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Модификатор доступа | Тип | Описание |
| tables | private | ArrayList<MyTable> | Объекты-таблицы текущего запуска |

Таблица 9 – Методы класса Session

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Модификатор доступа | Входные параметры | Тип | Описание |
| setTable | package-private | CreateTable createTable | void | Добавляет элемент в tables |
| setTable | package-private | CreateTable createTable, List relations | void | Добавляет элемент в tables |
| setTable | package-private | MyTable myTable | void | Добавляет элемент в tables |
| getTables | package-private | - | MyTable | Getter для tables |

*MyTable*

Модификатор доступа – package private.

Таблица 10 – Поля класса Session

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Модификатор доступа | Тип | Описание |
| table | private | Table | Объект, хранящий имя таблицы и схемы |
| columns | private | ArrayList<ColumnDefinition> | Множество столбцов таблицы, их типов и свойств |
| relations | private | List<String> | Список связей таблицы |

Таблица 11 – Методы класса Session

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Модификатор доступа | Входные параметры | Тип | Описание |
| setTable | package-private | Table table | void | Setter для tables |
| setColumns | package-private | ArrayList<ColumnDefinition> columns | void | Setter для columns |
| setRelations | package-private | List relations | void | Setter для relations |
| getTables | package-private | - | Table | Getter для tables |
| getColumns | package-private | - | ArrayList<ColumnDefinition> | Getter для columns |
| getRelations | package-private | - | List<String> | Getter для relations |

*Formatter*

Модификатор доступа – package private.

Таблица 12 – Поля класса Formatter

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Модификатор доступа | Тип | Описание |
| graph | private | StringBuilder | Хранит DOT запрос для создания графа |
| cfgProp | private | String | Путь к конфигурационному файлу |
| configFile | private | Properties | Конфигурация |
| TEMP\_DIR | private | String | Путь к директории для временных файлов |
| DOT | private | String | Путь к dot.exe |
| dpiSizes | private | int[] | Значения dpi для dot |
| currentDpiPos | private | int | Параметр Gdpi для dot |
| logger | private | Logger | Модуль логирования программы |

Таблица 13 – Методы класса Formatter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Модификатор доступа | Входные параметры | Тип | Описание |
| Formatter | package-private | - | - | Конструктор. Инициализирует модуль логирования |
| add | package-private | String line | void | Добавляет строку в graph |
| addln | package-private | String line | void | Добавляет строку и переход на новую строку в graph |
| addln | package-private | - | void | Добавляет переход на новую строку в graph |
| start\_graph | package-private | - | String | Добавляет начало dot запроса |
| end\_ graph | package-private | - | String | Добавляет завершение dot запроса |
| getDotSource | package-private | - | String | Возвращает dot запрос |
| getGraph | package-private | String dot\_source, String type | byte[] | Возвращает граф в виде массива байт |
| writeDotSourceToFile | private | String str | File | Запись dot запроса в файл |
| writeGraphToFile | package-private | byte[] img, File to | int | Запись графа в буфер |
| toDotFormat | package-private | Session thisSession | String | Построение dot запроса |
| tableToDotFormat | private | MyTable table | String | Представление таблицы в виде части dot запроса |
| relationToDotFormat | private | MyTable table | String | Представление отношения в виде части dot запроса |
| get\_img\_stream | private | File dot, String type | byte[] | Выполнение dot запроса |

***Разработка алгоритма***

Работу приложения можно условно разделить на три последовательных этапа.

Сначала происходит ввод исходных данных пользователем, затем данные последовательно считываются и обрабатываются, идет синтаксический анализ. Преобразованные на основе синтаксического анализа данные используются для построения графа. Алгоритмы синтаксического анализа и преобразования данных, а также построения графической модели описаны ниже.

Общий алгоритм графически представлен на рис. 9.



Рис. 9 Общий алгоритм программы

Процессы, обозначенные на рис. 9 как «Обработка стандартного DDL» и «Обработка скриптов DDL CTAS» представлены на рис. 10 и 11 соответственно.



Рис. 10 Алгоритм обработки стандартного DDL



Рис. 11 Алгоритм обработки скриптов DDL CTAS

*Алгоритм синтаксического анализа*

Функции синтаксического анализа:

* обработка массива SQL DDL запросов;
* извлечение из запросов информации о таблицах;
* представление этой информации в виде структуры объектов

Входные данные: массив объектов типа String. Каждый элемент хранит путь к текстовому файлу.

Выходные данные: объект типа Session, хранящий полученные из запросов таблицы, их связи и столбцы.

В основе синтаксического анализа в данной программе заложен метод сопоставления с образцом [13]. В качестве шаблона используются регулярные выражения.

Массив входных данных обрабатывается последовательно. Из массива извлекается очередная строка, считывается файл, лежащий по адресу из данной строки. Проверяется соответствие файла одному из поддерживаемых форматов SQL DDL запросов.

Сначала проверяется, соответствуют ли скрипт типу CREATE TABLE AS SELECT. Для этого используется регулярное выражение *^CREATE TABLE .\*(AS)?SELECT*. Поскольку запросы CTAS строятся на других таблицах, они преобразуются в последнюю очередь. Поэтому если при проверке найден такой скрипт, файл с этим скриптом копируется в буферную директорию, а путь к нему заносится в буферный массив. Если файл не относится к типу CREATE TABLE AS SELECT, выполняется дальнейшая проверка.

Проверяется соответствие типу CREATE TABLE. Для этого скрипт средствами JSqlParser преобразуется в объект типа Statement. Statement – класс, соответствующий SQL выражению, поддерживаемому фреймворком JSqlParser. От него наследуются классы, соответствующие конкретным типам запросов (CreateTable, Select, Truncate, Insert, Update и др.). Далее определяется, принадлежит ли созданный объект классу CreateTable с учетом наследования. При положительном результате проверки, выполняются дальнейшие преобразования. Если одно из преобразований не удалось, это означает, что проверяемый файл не удовлетворяет требованиям (не является SQL DDL скриптом, либо не соответствует поддерживаемому синтаксису).

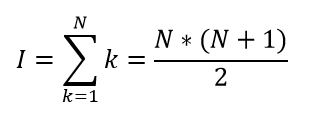
Для таблицы создается объект типа MyTable. Из объекта типа CreateTable извлекаются имя таблицы и схемы, названия и типы столбцов при помощи методов JSqlParser. Выполняется поиск внешних ключей по регулярным выражениям *(?<=FOREIGN KEY [(]).\** и *(?<=REFERENCES ).\*(?= )*. Полученной о таблице информацией заполняется объект типа MyTable. Массив всех объектов MyTable, созданных для текущего выполнения программы хранится в объекте типа Session.

После обработки входного массива выполняется проверка на пустоту буферного массива, который заполнялся ранее при обнаружении скриптов CREATE TABLE AS SELECT. Если массив не пуст, скрипты CTAS считываются и обрабатываются.

Для анализа и преобразования скрипта средствами JSqlParser скрипт типа CTAS форматируется. Название таблицы сохраняется в буферную переменную. Сам скрипт обрезается, оставляется только SELECT часть запроса, поскольку SELECT запросы легко преобразуются с помощью JSqlParser. Создается объект типа Select, в результате проверки его полей извлекаются названия столбцов, а также FROM и JOIN части запроса для определения связей таблицы. Выполняется поиск таблиц из FROM и JOIN предложений (это таблицы, на основе которых строится текущая). Если они найдены, запоминаются связи с ними. Типы данных для столбцов таблицы берутся из таблиц, на основе которых она построена. Для этого находится объект с таблицей, в нем находится столбец с соответствующим названием и извлекается его тип. Полученной о таблице информацией заполняется объект типа MyTable.

Стоит выделить проблему, возникающую при анализе CREATE TABLE AS SELECT скриптов. Таблицы, создаваемые с помощью CTAS, строятся на основе других таблиц, из которых делается выборка данных. При этом они могут строиться на основе таблиц, которые также создаются с помощью CTAS. Может возникнуть ситуация, когда скрипт для создания таблицы-родителя еще не обработан, а ее потомок обрабатывается в данный момент. Для такой таблицы будет невозможно обнаружить связи и типы данных столбцов.

Для решения этой проблемы цикл обработки скриптов CREATE TABLE AS SELECT организован особым образом. Скрипты заносятся в очередь. Скрипт извлекается из очереди, выполняется поиск таблиц-родителей. Если хотя бы одна таблица-родитель не обработана ранее (для нее не найден объект MyTable в Session), то скрипт снова заносится в конец очереди и будет проверен после всех остальных. В худшем случае при однократной проверке N скриптов типа CTAS должен быть обработан минимум 1 скрипт, для этого будет сделано N итераций. На втором кругу проверки в худшем случае будет обработан также 1 скрипт, при этом число итераций уменьшится до N-1. Таким образом, общее максимальное число итераций для успешного анализа всех N скриптов типа CTAS находится как N-я частичная сумма ряда натуральных чисел. Обозначим общее число итераций I. Для него справедлива следующая формула:



Цикл обработки CTAS скриптов проходит с учетом максимального числа итераций для текущей длины массива скриптов. Если лимит итераций достигнут, оставшиеся в очереди скрипты преобразуются, при этом их связи и типы данных столбцов (все, если не найдена ни одна таблица-родитель, либо только их часть) не будут определены. Неизвестные типы данных для таких таблиц будут иметь на визуальной схеме значение NULL.

*Алгоритм построения графической модели*

Входные данные: массив объектов, хранящих информацию о таблицах.

Выходные данные: визуальная схема базы данных.

Построение графической модели выполняется в GraphViz. Для этого сначала должен быть сформирован dot запрос для создания графа.

Информация о таблицах извлекается из MySession объектов, на ее основе собирается dot запрос.

Пример формируемого запроса:

***digraph G {***

***table1 [style=filled, fillcolor="#BFC9CA", shape=none, margin=0, label=<<TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0" CELLPADDING="4"><TR><TD COLSPAN="2">SCHEMA\_NAME.TABLE\_NAME</TD></TR><TR><TD>COLUMN\_1</TD><TD>COLUMN\_DATATYPE\_1</TD></TR><TR><TD>COLUMN\_2</TD><TD>COLUMN\_DATATYPE\_2</TD></TR></TABLE>>];***

***table1 [style=filled, fillcolor="#BFC9CA", shape=none, margin=0, label=<<TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0" CELLPADDING="4"><TR><TD COLSPAN="2">SCHEMA\_NAME.TABLE\_NAME</TD></TR><TR><TD>COLUMN\_1</TD><TD>COLUMN\_DATATYPE\_1</TD></TR><TR><TD>COLUMN\_2</TD><TD>COLUMN\_DATATYPE\_2</TD></TR></TABLE>>];***

***table1->table2;}***

Рис. 12 – Пример DOT запроса, формируемого в программе

На примере показан запрос для создания графической модели базы данных, состоящей из двух таблиц, имеющих между собой связь.

Как можно заметить из примера, стиль узлов dot графа может задаваться html кодом.

Как только запрос будет готов, он сохраняется в файл с расширением .dot. Затем консольной командой вызывается GraphViz и выполняется построение графа. Построенная графическая модель сохраняется в виде PDF файла. Файл открывается сразу после успешного завершения построения. PDF формат позволяет не терять качество схемы при ее масштабировании.

***Разработка пользовательского интерфейса***

Пользовательский интерфейс подразумевает только одну категорию пользователей и предназначен для упрощения ввода исходных данных.

Интерфейс содержит текстовое поле и две кнопки, отвечающие за открытие диалогового окна для выбора входных файлов и запуска построения схемы соответственно. Текстовое поле отображает входные данные – пути к файлам, содержащим SQL DDL скрипты.

В результате выполнения основного кода программы откроется построенная графическая модель, либо возникнет всплывающее окно с сообщением об ошибке, возникшей в ходе программы.

*Инструкция по использованию интерфейса*

Когда пользователь запускает приложение, он видит перед собой интерфейс, представленный на рис. 13.

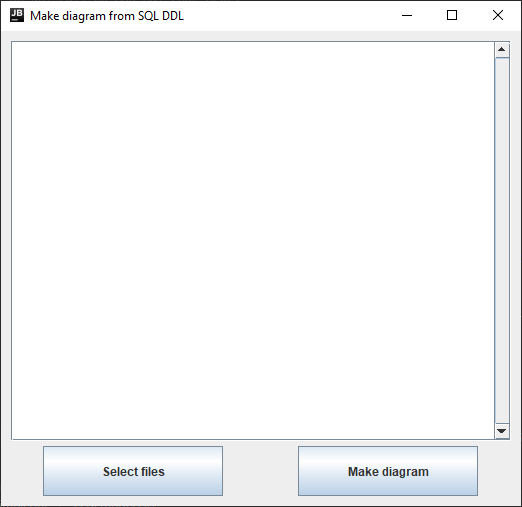


Рис. 13 Пользовательский интерфейс при его запуске

Сперва он должен ввести данные. Для этого необходимо нажать на кнопку «Select files». По нажатию кнопки открывается диалоговое окно для ввода файлов (рис. 14). В данном окне пользователь нажатием левой кнопки мыши (ЛКМ) должен выбрать файлы со скриптами, которые он хочет преобразовать, затем нажать «Open», либо «Cancel» для отмены ввода. При однократном вызове окна ввода, пользователь может добавить несколько файлов. Для этого выбирать их нужно при зажатой клавише Ctrl, либо, если файлы находятся в одной директории и лежат один за другим, при зажатой клавише Shift нажатием ЛКМ на первый в списке файл, затем на последний. В каждом файле должен лежать лишь 1 SQL DDL скрипт. Ввод файлов, содержащих более 1 скрипта, либо не соответствующих синтаксису SQL DDL приведет к ошибке в ходе работы программы.

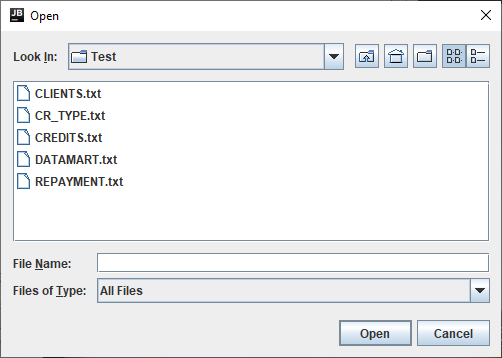


Рис. 14 Диалоговое окно выбора входных файлов

Все выбранные пользователем файлы будут отображаться в главном окне интерфейса (рис. 15). Убедившись, что все необходимые файлы с данными добавлены, пользователь может запустить построение графической модели, нажав на кнопку «Maker diagram». Построенная схема откроется в виде PDF документа.

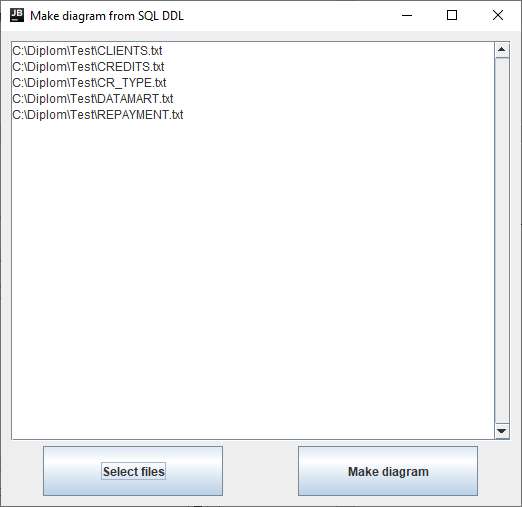


Рис. 15 Интерфейс после ввода данных

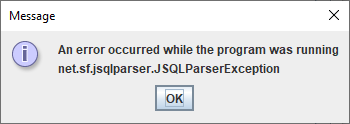


Рис. 16 Пример всплывающего окна при ошибке выполнения программы

**Тестирование**

Тестирование проводится по стратегии «черного ящика» - анализируется результат выполнения программы в зависимости от входных данных. Целью проведения тестирования приложения является демонстрация корректной работы при входных данных, удовлетворяющих требованиям, а также данных, не соответствующих им.

Ввод входных данных осуществляется пользователем через интерфейс приложения.

Порядок ввода данных следующий:

1. Пользователь открывает программу, перед ним открывается главное окно программы (рис. 13)
2. Необходимо нажать на кнопку «Select files»
3. В открывшемся диалоговом окне (рис. 14) открыть директорию, содержащую подготовленные DDL скрипты. Выбрать один или несколько (с зажатой shift или ctrl) файлов, затем нажать open.
4. Добавленные файлы будут отображаться в главном окне (рис. 15). Если добавлены не все необходимые файлы со скриптами, повторить действия 2-3.
5. Как только ввод окончен, нажать на копку «Make diagram». Запустится процесс обработки скриптов. Как только скрипты будут обработаны и визуальная модель построится, на ПК пользователя появится PDF файл diagram.pdf, этот файл откроется сразу по окончанию его создания.

***Успешное выполнение программы***

В качестве источника входных данных используется база данных компании «Кадровый учет».

Описания таблиц:

* REGIONS - содержит строки, которые представляют регион, такой как Америка, Азия и так далее.
* COUNTRIES - содержит строки для стран, каждая из которых связывается с областью.
* LOCATIONS - содержит определенный адрес определенного офиса, склада или производственного участка компании в определенной стране.
* DEPARTMENTS - показывает подробные данные об отделах, в которых работают сотрудники. У каждого отдела может быть отношение, представляющее начальника отдела в таблице EMPLOYEES.
* EMPLOYEES - содержит подробные данные о каждом сотруднике, работающем в отделе. Некоторые сотрудники могут быть не присвоены никакому отделу.
* JOB - содержит должности, которые могут быть закреплены за каждым сотрудником.
* JOB\_HISTORY - содержит историю изменения должностей сотрудников. Если сотрудник изменяет отдел в пределах задания или меняет работу в пределах отдела, новая строка вставляется в эту таблицу с более ранней информацией о задании сотрудника.

База данных хранится в Oracle Database. Для того, чтобы ввести информацию о ней в программу, подготовлен набор DDL скриптов, описывающих создание таблиц, а также их связи. Информация о каждой таблице и ее связях представлена в отдельном скрипте с расширением .ddl. Ниже приведен листинг данных скриптов.

/\* Регионы \*/

CREATE TABLE regions (

/\* Уникальный идентификатор региона \*/

region\_id NUMBER(10) NOT NULL,

/\* Название региона \*/

region\_name VARCHAR2(50)

);

/\* Определение первичного ключа таблицы \*/

ALTER TABLE regions ADD CONSTRAINT regions\_pk PRIMARY KEY ( region\_id );

Рис. 17 Regions.ddl

/\* Страны \*/

CREATE TABLE countries (

/\* Уникальный идентификатор страны \*/

country\_id NUMBER(10) NOT NULL,

/\* Название страны \*/

country\_name VARCHAR2(30),

/\* Идентификатор региона, к которому относится страна \*/

region\_id NUMBER(10) NOT NULL

);

/\* Определение первичного ключа таблицы \*/

ALTER TABLE countries ADD CONSTRAINT countries\_pk PRIMARY KEY ( country\_id );

/\* Определение внешнего ключа таблицы к таблице регионов\*/

ALTER TABLE countries

ADD CONSTRAINT countries\_regions\_fk FOREIGN KEY ( region\_id )

REFERENCES regions ( region\_id )

NOT DEFERRABLE;

Рис. 18 Countries.ddl

/\* Адрес \*/

CREATE TABLE locations (

/\* Уникальный идентификатор адреса \*/

location\_id NUMBER(10) NOT NULL,

/\* Улица \*/

street\_address VARCHAR2(100),

/\* Индекс \*/

postal\_code NUMBER(10),

/\* Город \*/

city VARCHAR2(100),

/\* Регион страны \*/

state\_province VARCHAR2(100),

/\* Идентификатор страны \*/

country\_id NUMBER(10) NOT NULL

);

/\* Определение первичного ключа таблицы \*/

ALTER TABLE locations ADD CONSTRAINT locations\_pk PRIMARY KEY ( location\_id );

/\* Определение внешнего ключа таблицы к таблице стран \*/

ALTER TABLE locations

ADD CONSTRAINT locations\_countries\_fk FOREIGN KEY ( country\_id )

REFERENCES countries ( country\_id )

NOT DEFERRABLE;

Рис. 19 Locations.ddl

/\* Подразделения \*/

CREATE TABLE departments (

/\* Уникальный идентификатор подразделения \*/

department\_id NUMBER(10) NOT NULL,

/\* Название подразделения \*/

department\_name VARCHAR2(100),

/\* Руководитель подразделения \*/

manager\_id NUMBER(10),

/\* Адрес подразделения \*/

location\_id NUMBER(10) NOT NULL

);

/\* Определение первичного ключа таблицы \*/

ALTER TABLE departments ADD CONSTRAINT departmens\_pk PRIMARY KEY ( department\_id );

/\* Определение внешнего ключа таблицы к таблице адресов \*/

ALTER TABLE departments

ADD CONSTRAINT departmens\_locations\_fk FOREIGN KEY ( location\_id )

REFERENCES locations ( location\_id )

NOT DEFERRABLE;

Рис. 20 Departments.ddl

/\* Сотрудники \*/

CREATE TABLE employees (

/\* Уникальный идентификатор сотрудника \*/

employee\_id NUMBER(10) NOT NULL,

/\* Имя сотрудника \*/

first\_name VARCHAR2(50),

/\* Фамилия идентификатор сотрудника \*/

last\_name VARCHAR2(50),

/\* Электронная почта сотрудника \*/

email VARCHAR2(50),

/\* Мобильный телефон сотрудника \*/

phone\_number VARCHAR2(20),

/\* Дата найма сотрудника \*/

hire\_date DATE NOT NULL,

/\* Идентификатор должности сотрудника \*/

job\_id NUMBER(10) NOT NULL,

/\* Зарплата сотрудника \*/

salary VARCHAR2(30),

/\* Ставка сотрудника \*/

comission\_pct VARCHAR2(30),

/\* Идентификатор руководителя сотрудника \*/

manager\_id NUMBER(10),

/\* Идентификатор подразделения сотрудника \*/

department\_id NUMBER(10) NOT NULL

);

/\* Определение первичного ключа таблицы \*/

ALTER TABLE employees ADD CONSTRAINT employees\_pk PRIMARY KEY ( employee\_id );

/\* Определение внешнего ключа таблицы к таблице подразделений \*/

ALTER TABLE employees

ADD CONSTRAINT employees\_departmens\_fk FOREIGN KEY ( department\_id )

REFERENCES departments ( department\_id )

NOT DEFERRABLE;

/\* Определение внешнего ключа таблицы к таблице истории изменения должностей \*/

ALTER TABLE employees

ADD CONSTRAINT employees\_job\_history\_fk FOREIGN KEY ( employee\_id,

hire\_date )

REFERENCES job\_history ( employee\_id,

start\_date )

NOT DEFERRABLE;

/\* Определение внешнего ключа таблицы к таблице должностей \*/

ALTER TABLE employees

ADD CONSTRAINT employees\_jobs\_fk FOREIGN KEY ( job\_id )

REFERENCES jobs ( job\_id )

NOT DEFERRABLE;

Рис. 21 Employees.ddl

/\* Должности \*/

CREATE TABLE jobs (

/\* Уникальный идентификатор должности \*/

job\_id NUMBER(10) NOT NULL,

/\* Название должности \*/

job\_title VARCHAR2(100),

/\* Минимальная зарплата \*/

min\_salary VARCHAR2(30),

/\* Максимальная зарплата \*/

max\_salary VARCHAR2(30)

);

/\* Определение первичного ключа таблицы \*/

ALTER TABLE jobs ADD CONSTRAINT jobs\_pk PRIMARY KEY ( job\_id );

Рис. 22 Job.ddl

/\* История изменения должностей \*/

CREATE TABLE job\_history (

/\* Уникальный идентификатор сотрудника \*/

employee\_id NUMBER(10) NOT NULL,

/\* Дата начала работы на должности \*/

start\_date DATE NOT NULL,

/\* Дата окончания работы на должности \*/

end\_date DATE,

/\* Уникальный идентификатор должности \*/

job\_id NUMBER(10) NOT NULL,

/\* Уникальный идентификатор подразделения \*/

department\_id NUMBER(10) NOT NULL

);

/\* Определение первичного ключа таблицы \*/

ALTER TABLE job\_history ADD CONSTRAINT job\_history\_pk PRIMARY KEY ( employee\_id,

start\_date );

/\* Определение внешнего ключа таблицы к таблице подразделений \*/

ALTER TABLE job\_history

ADD CONSTRAINT job\_history\_departmens\_fk FOREIGN KEY ( department\_id )

REFERENCES departments ( department\_id )

NOT DEFERRABLE;

/\* Определение внешнего ключа таблицы к таблице должностей \*/

ALTER TABLE job\_history

ADD CONSTRAINT job\_history\_jobs\_fk FOREIGN KEY ( job\_id )

REFERENCES jobs ( job\_id )

NOT DEFERRABLE;

Рис. 23 Job\_history.ddl

Также для тестирования разработаны скрипты для создания таблиц – срезов данных. DDL скрипты этих таблиц имеют тип CREATE TABLE AS SELECT.

В тестировании проверяется работа приложения на двух срезах данных. Первый отображает сотрудников, работающих только в главном офиса. Второй отображает регионы, где сотрудники компании имеют зарплату, ниже средне допустимой для их должности.

Описания таблиц:

* EMPLOYEES\_OF\_MAIN\_OFFICE – показывает сотрудников, работающих в главном офисе.
* EMPLOYEES\_OF\_REGION – показывает всех сотрудников и регионы, в которых они работают.
* EMPLOYEES\_UNDER\_AVG\_SALARY – показывает регионы, в которых сотрудники имеют зарплату ниже средне допустимой для их должности.

/\* Сотрудники, работающие в главном офисе \*/

CREATE TABLE employees\_of\_main\_office AS (

/\* Название подразделения \*/

SELECT departments.department\_name as department\_name,

/\* Имя сотрудника \*/

employees.first\_name as first\_name,

/\* Фамилия сотрудника \*/

employees.last\_name as last\_name

FROM employees join departments on employees.department\_id = departments.department\_id

join locations on departments.location\_id = locations.location\_id

/\* Адрес главного офиса \*/

WHERE locations.street\_address LIKE 'MOSCOW, TVERSKAYA ST., 8/2'

);

Рис. 24 Employees\_of\_main\_office.ddl

/\* Сотрудники и регионы, в которых они работают \*/

CREATE TABLE employees\_of\_region AS (

/\* Идентификатор сотрудника \*/

SELECT employees.employee\_id as employee\_id,

/\* Имя сотрудника \*/

employees.first\_name as first\_name,

/\* Фамилия сотрудника \*/

employees.last\_name as last\_name,

/\* Идентификатор региона \*/

regions.region\_id as region\_id,

/\* Название региона \*/

regions.region\_name as region\_name,

/\* Идентификатор должности \*/

employees.job\_id,

/\* Зарплата сотрудника \*/

employees.salary

FROM employees join departments on employees.department\_id = departments.department\_id

join locations on departments.location\_id = locations.location\_id

join countries on locations.country\_id=countries.country\_id

join regions on countries.region\_id=regions.region\_id

);

Рис. 25 Employees\_of\_region.ddl

/\* Регионы, в которых работают сотрудники, зарплата которых ниже средне допустимой на данной должности \*/

CREATE TABLE employees\_under\_avg\_salary AS (

/\* Название региона \*/

SELECT employees\_of\_region.region\_name as region\_name,

employees\_of\_region.employee\_id as employee\_id,

/\* Имя сотрудника \*/

employees\_of\_region.first\_name as first\_name,

/\* Фамилия сотрудника \*/

employees\_of\_region.last\_name as last\_name,

/\* Идентификатор должности \*/

jobs.job\_id as job\_id,

/\* Название должности \*/

jobs.job\_title as job\_title FROM employees\_of\_region join jobs on employees\_of\_region.job\_id=jobs.job\_id

WHERE ((job\_id.max\_salary + job\_id.min\_salary)/2) > employees\_of\_region.salary

);

Рис. 26 Employees\_under\_avg\_salary.ddl

Ожидаемый результат: Программа успешно обработает массив скриптов и сформирует графическую схему с учетом особенностей языка (типов данных, ключевых слов). Схема имеет легко читаемый вид.

Результат:

Программой сгенерирован файл DIAGRAM.pdf, содержащий построенную визуальную модель (рис. 27). Все таблицы, их атрибуты, а также связи корректны, следовательно, тестирование выполнено успешно.

Данная модель наглядно демонстрирует организацию таблиц и их связей в базе данных и пригодна для дальнейшего анализа базы данных.

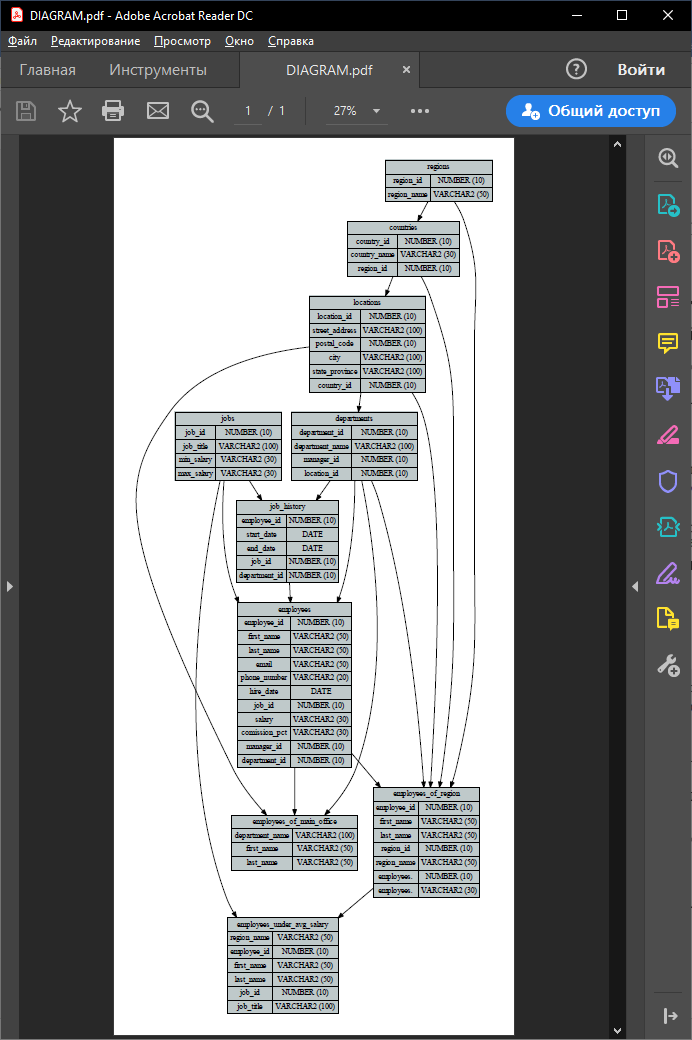


Рис. 27 Визуальная модель, полученная в результате теста 1

На рис. 28 и 29 представлены участки данной диаграммы в увеличенном формате.

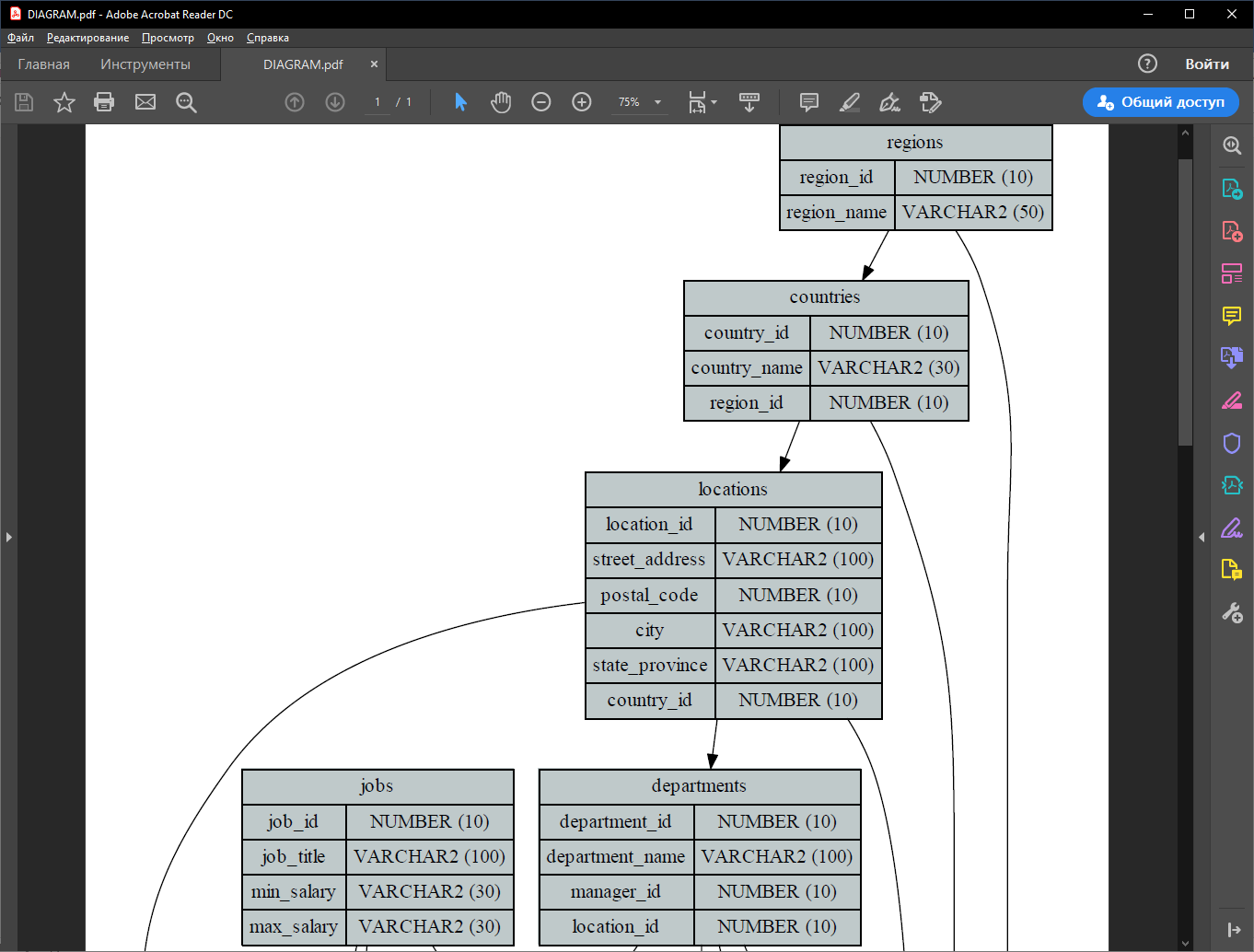


Рис. 28 Верхний участок схемы

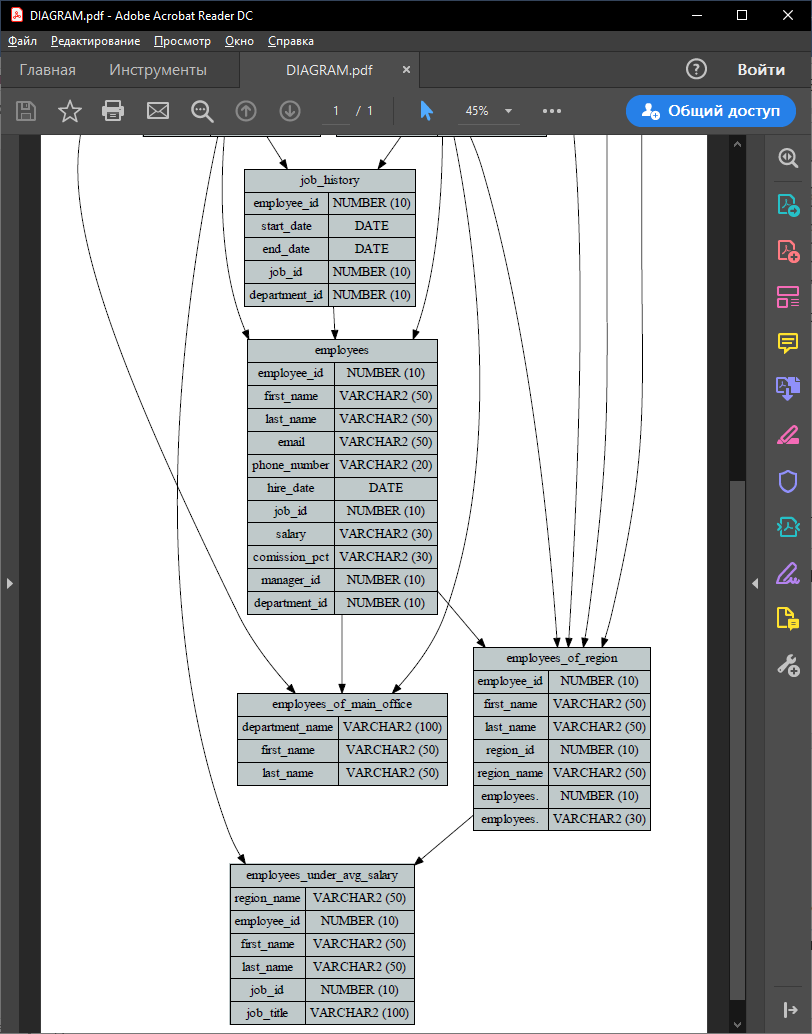


Рис. 29 Нижний участок схемы

***Выполнение программы при ошибке в данных***

*Неподдерживаемый формат SQL*

Для данного теста в предыдущий набор скриптов внесены изменения: вместо CTAS скрипта создания таблицы FILMS\_OF\_ACTOR имеется SELECT запрос.

SELECT employees\_of\_region.region\_name as region\_name,

employees\_of\_region.employee\_id as employee\_id,

/\* Имя сотрудника \*/

employees\_of\_region.first\_name as first\_name,

/\* Фамилия сотрудника \*/

employees\_of\_region.last\_name as last\_name,

/\* Идентификатор должности \*/

jobs.job\_id as job\_id,

/\* Название должности \*/

jobs.job\_title as job\_title FROM employees\_of\_region join jobs on employees\_of\_region.job\_id=jobs.job\_id

WHERE ((job\_id.max\_salary + job\_id.min\_salary)/2) > employees\_of\_region.salary

;

Рис. 30 Измененный скрипт

Ожидаемый результат: всплывающее окно с ошибкой.

Результат:

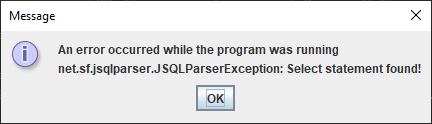


Рис. 31 – Результат выполнения программы при входном Select запросе

Полученный результат показывает, что при распознавании SQL запроса типа SELECT, программа сообщает об ошибке и прекращает свою работу, что соответствует ожидаемому результату.

*Формат входных файлов, не соответствующий SQL*

Входные данные: файл с данными в формате XML.

Ожидаемый результат: всплывающее окно с ошибкой.

Результат:

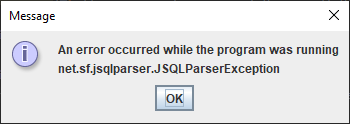


Рис. 32 – Результат выполнения программы при некорректном формате входных данных

Полученный результат показывает, что в случае, когда содержимое файла не соответствует SQL DDL, программа не распознает в нем ни один из поддерживаемых форматов и сообщает об ошибке.

**Заключение**

В результате разработки исследован метод визуализации данных для повышения производительности анализа баз данных и проектирования срезов данных. Были проанализированы инструменты для визуализации данных.

На основе собранного материала было разработано приложение, преобразующее набор SQL DDL скриптов в визуальную модель базы данных. Произведен выбор среды программирования и разработан пользовательский интерфейс, обеспечивающий удобный ввод данных. Разработаны алгоритмы синтаксического анализа скриптов и визуализации полученной в результате синтаксического анализа информации, а также код, реализующий эти алгоритмы в программе. Общий объем исходного кода программы – 1084 строки.

Проведено тестирование программы на массиве SQL скриптов для создания таблиц, которые преобразуются программой в наглядную графическую модель базы данных.

Данное приложение может применяться разработчиками баз данных при необходимости анализа таблиц существующей базы данных с целью ее доработки.

**Список используемой литературы**

1. Брила, Б. Oracle Database 11g. Настольная книга администратора баз данных / Б. Брила, К. Луни. – Litres, 2019. – 848 с.
2. Левчук, Е. А. Технологии организации хранения и обработки данных: учеб. пособие / Е. А. Левчук. – 3-е изд. – Минск: Выш. шк., 2007. – 239 с.
3. Oracle SQL Developer: [сайт]. URL: <https://docs.oracle.com/en/database/oracle/sql-developer/index.html>
4. DBMS Tools: [сайт]. URL: <https://dbmstools.com/categories/database-diagram-tools>
5. Dataedo: [сайт]. URL: <https://dataedo.com/product>
6. SchemaSpy: [сайт]. URL: <https://schemaspy.readthedocs.io/en/latest/>
7. DataGrip: [сайт]. URL: <https://www.jetbrains.com/ru-ru/datagrip/features/>
8. Navicat Premium: [сайт]. URL: <https://www.navicat.com/en/products/navicat-premium>
9. GraphViz: [сайт]. URL: <https://www.graphviz.org/>
10. JSqlParser: [сайт]. URL: <http://jsqlparser.sourceforge.net/home.php>
11. Давыдов С. В. IntelliJ IDEA. Профессиональное программирование на Java / С. В. Давыдов – БХВ-Петербург, 2005. – 800 с.
12. Ian F. Darwin, Java Cookbook, Fourth Edition. [O'Reilly Media](https://learning.oreilly.com/library/publisher/oreilly-media-inc/?utm_medium=referral&utm_campaign=publisher&utm_source=oreilly&utm_content=catalog&utm_content=catalog), 2019, 614p.
13. Joshua Bloch, Effective Java, 3rd Edition. Addison-Wesley Professional, December 2017, 377p.