А.В. Цуканов, Н.А. Русина

Поиск зависимостей (Data Mining) в больших базах данных

Лабораторный практикум

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт финансов, экономики и управления

А.В. Цуканов, Н.А. Русина

Поиск зависимостей (Data Mining) в больших базах данных

Лабораторный практикум

Севастополь
СевГУ
2025

УДК 005.31:004.6(076.5)
ББК 65.23я73
Ц850

Р е ц е н з е н т ы:
А.И. Бохонский – д-р техн. наук, профессор кафедры «Цифровое проектирование», Севастопольский государственный университет
С.М. Братан – д-р техн. наук, заведующий кафедрой «Технология машиностроения», Севастопольский государственный университет

Цуканов, А.В.

Ц850 Поиск зависимостей (Data Mining) в больших базах данных: лабораторный практикум / А.В. Цуканов, Н.А. Русина ; Севастопольский государственный университет, Институт финансов, экономики и управления. – Севастополь : СевГУ, 2025. – 155 с.: ил. – Текст : электронный.

Содержание лабораторного практикума соответствует утвержденным в СевГУ рабочим программам дисциплин «Дейта майнинг в управлении» и «Диджитал-аналитика» по направлению подготовки 38.04.02 «Менеджмент».

Предназначен для студентов всех форм обучения и слушателей курсов повышения квалификации в рамках дополнительного профессионального образования.

УДК 005.31:004.6(076.5)
ББК 65.23я73

Рассмотрено и рекомендовано к изданию на заседании кафедры «Менеджмент и бизнес-аналитика» Института финансов, экономики и управления Севастопольского государственного университета, протокол № 8 от 10 июня 2024 г.

Содержание

Список сокращений 5
Предисловие 6
Введение 7
Глава 1. Интерфейсы системы управления большими базами данных Vertica 9
1.1 Начало работы с системой Vertica 9
1.2 Главное меню консоли менеджера 12
1.3 Область «Provision» 13
1.4 Область «Manage» 15
1.5 Область консоли «MC Tools» 25
1.6 Область «Recent Databases» 28
1.7 Выход из системы 29
1.8 Порядок выполнения работы 30
1.9 Содержание отчета 30
1.10 Контрольные вопросы 30
Глава 2. Создание базы данных в системе Vertica 31
2.1 Кластеры системы Vertica 31
2.2 Создание новой базы данных в системе Vertica 32
2.3 Создание новых отношений/таблиц базы данных 41
2.4 Операторы ALTER TABLE и DROP 50
2.5 Порядок выполнения работы 52
2.6 Содержание отчета 52
2.7 Контрольные вопросы 52
2.8 Варианты задания 52
Глава 3. Лабораторный практикум. Загрузка данных в системе Vertica 54
3.1 Оператор языка SQL «INSERT» 54
3.2 Оператор языка SQL “UPDATE” 58
3.3 Оператор языка SQL «DELETE» 60
3.4 Транзакции в системе Vertica 60
3.5 Порядок выполнения работы 62
3.6 Содержание отчета 62
3.7 Контрольные вопросы 63
3.8 Варианты задания 63
Глава 4. Запросы на получение данных в системе Vertica 66
4.1 Простые формы оператора SELECT языка SQL 66
4.2 Формирование условия выборки дан 72
4.3. Запросы с соединением таблиц 76
4.4 Подзапросы 79
4.5 Порядок выполнения работы 83
4.6 Содержание отчета 83
4.7 Контрольные вопросы 83
4.8 Варианты задания 84

Глава 5. Построение математических моделей 85
5.1 Классический регрессионный анализ 85
5.2 Построение регрессионных моделей для больших данных 87
5.3. Методы решения задачи оптимизации 88
5.4 Линейная регрессионная модель в системе Vertica 89
5.5 Работа в командной консоли администратора 90
5.6 Пример построения регрессионной модели 101
5.7 Порядок выполнения работы 109
5.8 Содержание отчета 111
5.9 Контрольные вопросы 111
5.10 Варианты задания 111
Список литературы 113
Приложение А. Установка и настройка системы VIRTUALBOX 115
Приложение Б. Демонстрационная база данных Vertica 137
Приложение В. Синтаксис основных предложений языка SQL 148

Список сокращений

ПО – программное обеспечение.
СУББД – система управления большими базами данных.
ОС – операционная система.
ПК – персональный компьютер.
SQL – стандартный структурированный язык запросов к базе данных.

Предисловие

Формирование четвертого технологического уклада тесно связано с цифровой экономикой, важность которой отмечается в Стратегии развития информационного общества в Российской Федерации на 2017 - 2030 годы [1].

Практикум предназначен для получения студентами навыков работы с большими базами данных и с методами поиска аналитических зависимостей в больших базах данных. Практикум содержит необходимые сведения из теории больших баз данных, методов статистического анализа информации и инструкции по использованию программного обеспечения.

Практикум также детально знакомит будущих менеджеров, специализирующихся в области бизнес-информатики, с технологией автоматизированного управления предприятием на основе современных информационных технологий.

В практикуме для выполнения лабораторных работ используется свободно распространяемое программное обеспечение для управления большими базами данных СУББД Vertica. При этом студенты осваивают технологию виртуальных вычислительных машин и основы работы с операционной системой LINUX [11].

Практикум содержит примеры моделирования и аналитики больших баз данных и индивидуальные задания для практического менеджмента больших баз данных.

Материал соответствует рабочим программам дисциплин «Data Mining в управлении», «Data Mining в экономике», «Диджитал аналитика», «Большие базы данных и социальные сети», «Бизнес аналитика» для подготовки студентов по направлению «Менеджмент». Лабораторный практикум также может быть использован для обучения студентов по дисциплинам «Автоматизированные системы управления предприятием», «Автоматизированные системы управления требованиями», «Цифровые технологии в бизнесе» и «Цифровые технологии в предпринимательстве».

Лабораторный практикум был использован в учебном процессе Севастопольского государственного университета и отработан при проведении лабораторных работ в течении нескольких лет.

Авторы благодарны рецензентам профессорам Севастопольского государственного университета А.И. Бохонскому и С.М. Братану за ценные замечания, сделанные при рецензировании лабораторного практикума.

Введение

Целью лабораторного практикума является получение знаний и практических навыков студентами, специализирующимися в области применения информационных технологий в менеджменте, для создания систем автоматизации управления предприятием, аналитической работы с большими базами данных и принятия эффективных решений в менеджменте.

При выполнении практикума студенты получают знания и навыки как в анализе и информационном моделировании корпоративных данных, описании текущих характеристик предприятия, так и в использовании методов Data Mining для анализа деятельности предприятия.

В качестве лабораторного оборудования используется система управления большими базами данных Vertica, которая позволяет решать следующие задачи менеджмента данных:

- Оптимизация хранения данных. Используются столбчатые хранилища данных, которые обеспечивают значительный прирост производительности бизнес-процессов, когда выполняется аналитическая обработка данных.
- Управление уровнями доступа, настройка и контроль СУББД Vertica с минимальным администрированием. Осуществление информационного менеджмента с целью защиты персональных данных и иной конфиденциальной информации от несанкционированного доступа.
- Выполнение загрузки данных и запросов в режиме реального времени, с высоким параллелизмом запросов и возможностью одновременной загрузки новых данных в систему и выполнения запросов к ним.
- Поддержка структурированных и неструктурированных данных. В дополнение к традиционным структурированным таблицам базы данных Vertica предоставляет гибкие таблицы, которые позволяют загружать и анализировать неструктурированные и полуструктурированные данные.
- Конструирование информационной модели данных.
- Поддержка стандартного языка запросов SQL.
- Проведение расширенной аналитики базы данных, включая машинное обучение, регрессионный анализ, классификацию, кластеризацию, геопространственную аналитику и аналитику временных рядов.

Индивидуальные задания для студентов содержат примеры из разных областей менеджмента и для различных предприятий.

Структурно учебное пособие состоит из пяти глав и приложений.

В первой главе рассматривается практическая работа по загрузке системы, изучается интерфейс системы, осваивается консоль менеджера базы данных.

Во второй главе студентам предлагается создать свою новую базу данных.

В третьей главе рассматривается и выполняется студентами в ходе практикума загрузка данных.

В четвертой главе студенты получают основные навыки по выполнению запросов в языке SQL.

В пятой главе изучаются и выполняются практические задания по использованию методов аналитики в системе Vertica.

Глава 1. Лабораторный практикум. Интерфейсы системы управления большими базами данных Vertica

Система Vertica [10] является одной и наиболее успешной системой управления большми базами данных, которая нашла применение для различного рода предприятий. В лабораторном практикуме используется бесплатная версия системы, реализованной в виртуальной машине, которая работает под управлением ОС Linux [11]. Процесс установки системы описан подробно в приложении А учебного пособия.

1.1 Начало работы с системой Vertica

Работа системы начинается с запуска монитора виртуальных машин, как показано на рис. 1.1. Для этого надо щелкнуть левой кнопкой мыши в списке виртуальных машин на машине с именем “vm”.

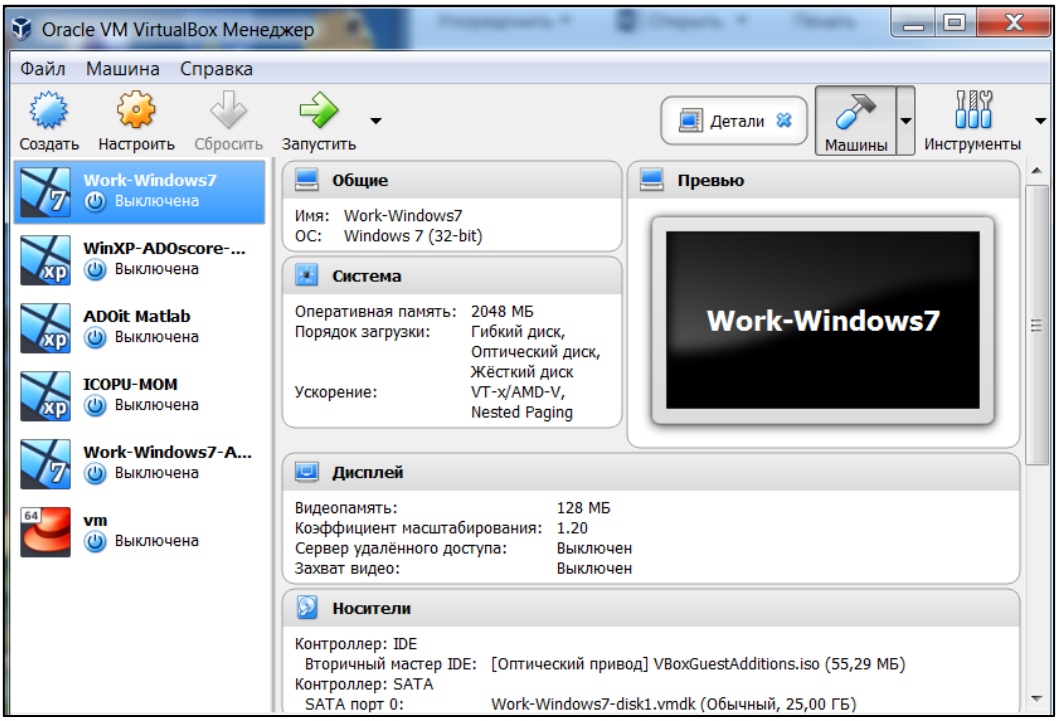


Рисунок 1.1 – Запуск виртуальной машины с загруженной системой

При запуске появится сообщение, показанное на рис. 1.2 о том, что клавиатура и мышь будут захвачены виртуальной машиной, как только Вы щелкните мышкой на экране виртуальной машины.

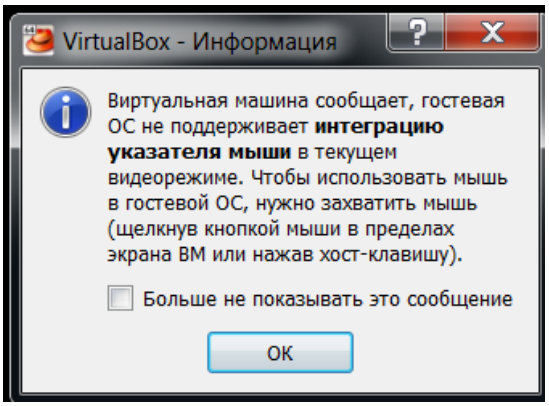


Рисунок 1.2 – Сообщение о перехвате управления мышью

Для возврата управления основной машине надо нажать правую клавишу *Control* на клавиатуре.

При загрузке операционной системы Linux на экран выводится список пользователей системы, как показано на рис. 1.3.

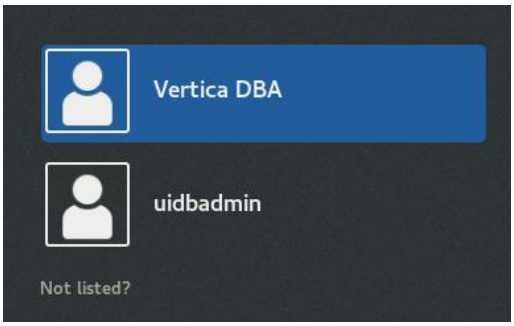


Рисунок 1.3 – Пользователи системы

Для системы первоначально созданы два пользователя. Оба пользователя могут быть администраторами системы, менеджерами или аналитиками базы данных. Пользователь под именем Vertica DBA работает с графическим интерфейсом системы, а пользователь под именем uidbadmin в командном режиме системы LINUX. Пароль входа в систему: **password**.

При входе на консоль менеджера открывается окно с основным меню системы LINUX, показанное на рис. 1.4.

Рисунок 1.4 – Рабочее поле системы LINUX

Если зайти в пункт “Application” (Приложения), то откроется окно с загруженными программами системы LINUX, показанное на рис. 1.5.

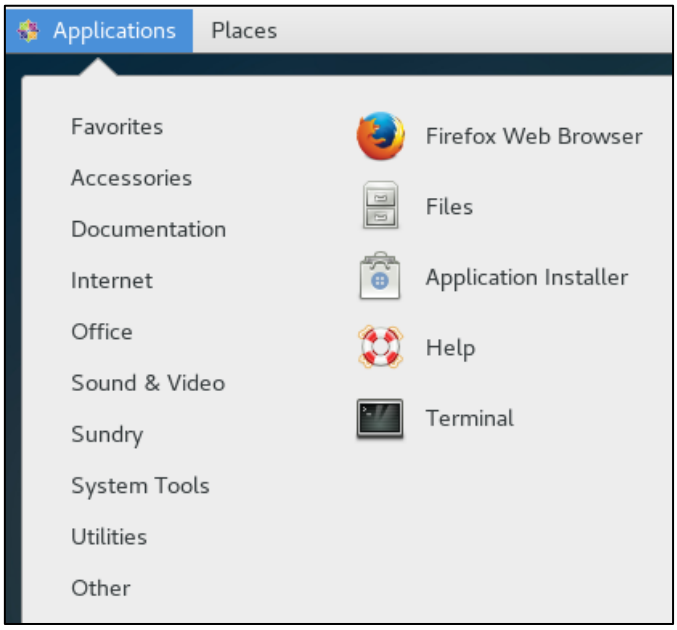


Рисунок 1.5 – Меню приложений

Для входа в консоль менеджера надо выбрать иконку на рабочем поле с именем “Management Console”. В результате откроется окно браузера интернета Mozilla Firefox с окном входа в систему Vertica, как показано на рис. 1.6.

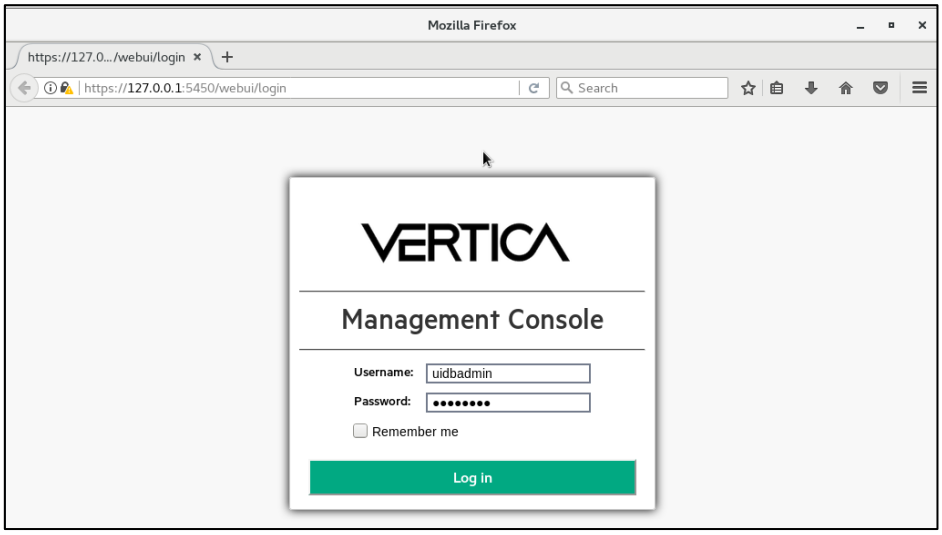


Рисунок 1.6 – Вход в систему

Для входа в систему для выполнения практикума используется тот же пароль, что и для входа в LINUX. Обычно по умолчанию он уже вставлен в поле пароля. Графический интерфейс системы создан в виде сайта, что упрощает работу с системой пользователям непрограммистам. Адрес входа задан в командной строке браузера. Если он там отсутствует, что может быть при другом входе в систему, то его надо набрать вручную.

1.2 Главное меню консоли менеджера

При входе в систему открывается доступ к главному меню системы. Главное меню состоит из пяти областей. Первые две области показаны на рис. 1.7.

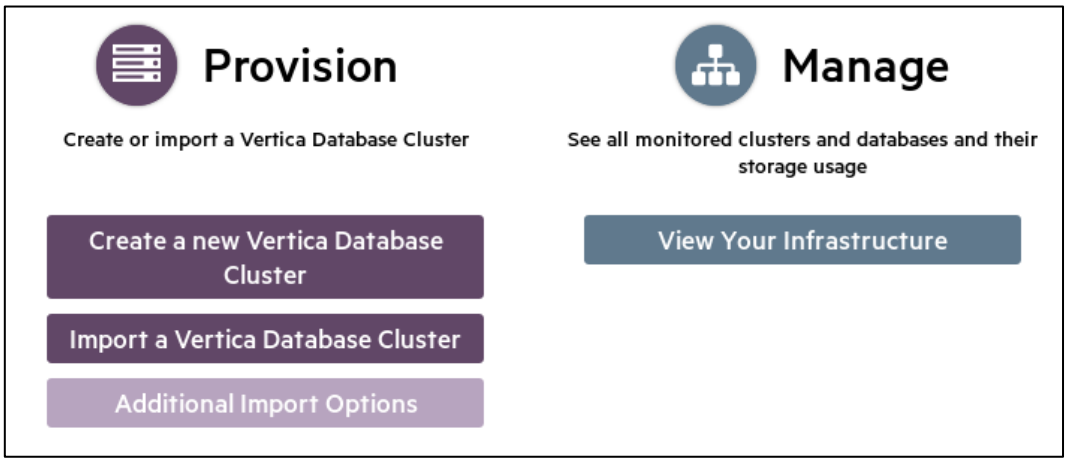


Рисунок 1.7 – Первые две области главного меню консоли менеджера

Область “**Provision**” (Обеспечение) осуществляет вход в средства создания и импорта кластера. Система Vertica может работать на нескольких компьютерах, которые логически объединяются в один кластер.

Область “**Manage**” (Управление информацией) позволяет просматривать информацию о кластерах, базах данных и процессах их использования.

На рис. 1.8 показана третья и четвертая области главного меню системы.

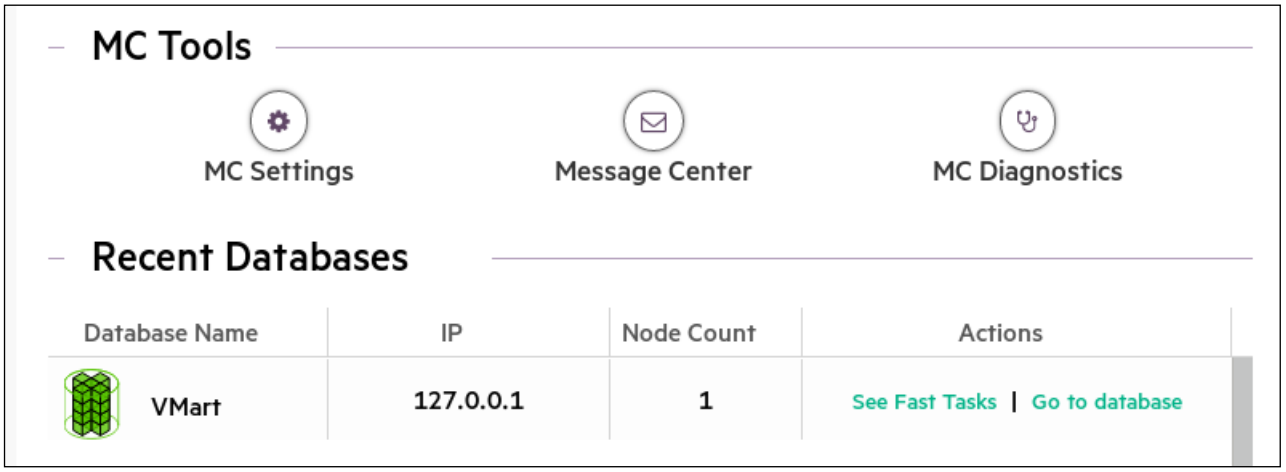


Рисунок 1.8 – Третья и четвертая области главного меню

Область “**MC Tools**” содержит три кнопки для работы с инструментами консоли менеджера. Область “**Recent Databases**” содержит информацию об имеющихся базах данных и их состоянии.

В низу страницы, как показано на рис. 1.9, содержится область со ссылками на сайты со справочной информацией о системе Vertica. Естественно, чтоб воспользоваться этой информацией должен быть доступ к интернету.

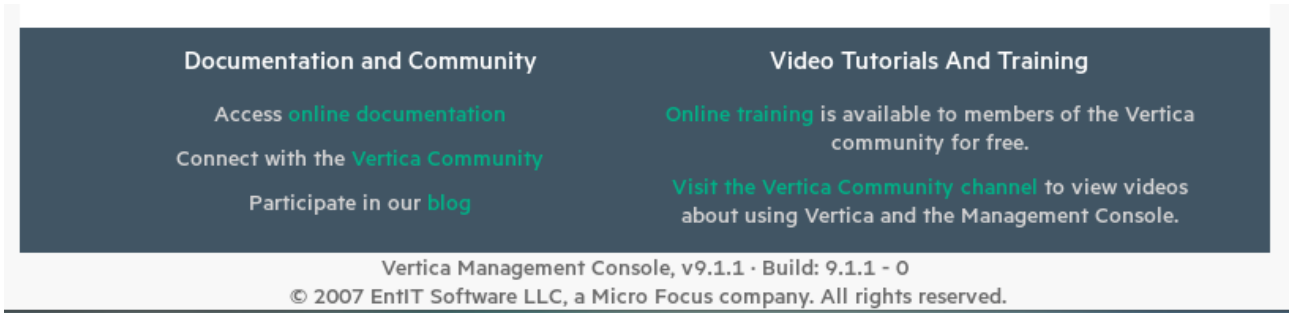


Рисунок 1.9 – Ссылки на справочную информацию

1.3 Область “Provision”

В этой области имеется три пункта меню.

Пункт меню “**Create a new Vertica Database Cluster**” открывает окно, показанное на рис. 1.10.

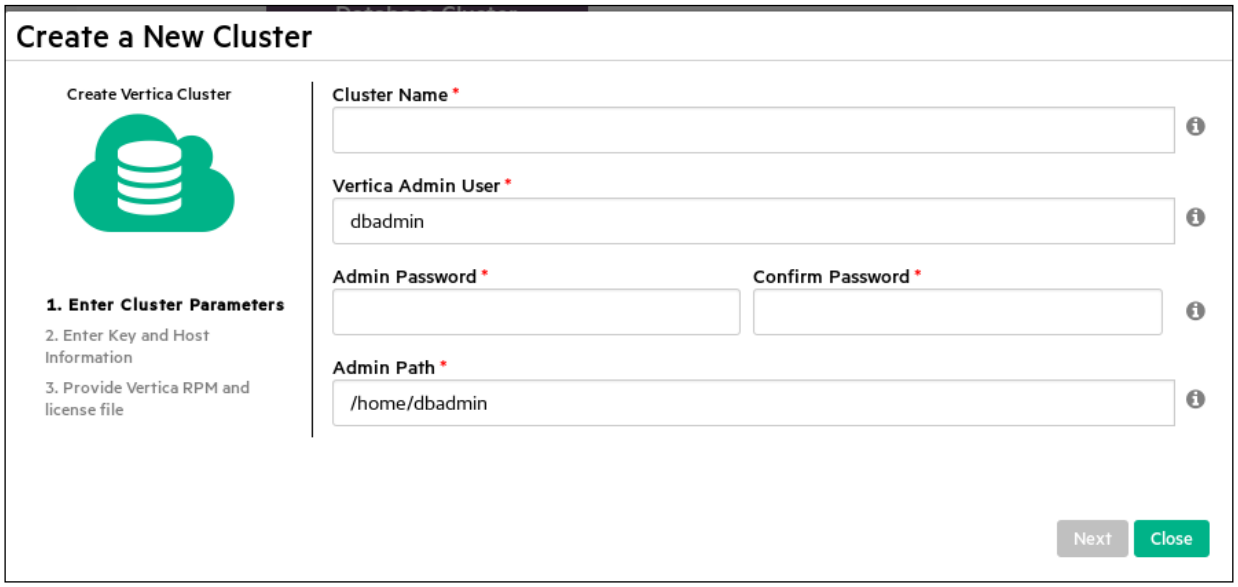


Рисунок 1.10 – Окно для создания нового кластера

В этом окне имеется подменю с тремя пунктами.

В первом пункте “**Enter Cluster Parameters**”, открытом по умолчанию на рис. 1.10, задаются основные параметры нового кластера.

Во втором пункте “**Enter Key and Host Information**” задается ключ для работы с кластером и информация о сервере, на котором находится доступ к кластеру.

В третьем пункте “**Provide Vertica RPM and license file**” необходимо указать файл с лицензией для работы с системой. Следует учитывать, что в бесплатной версии системы имеются ограничения для создания большой базы данных.

Пункт меню области “**Import a Vertica Database Cluster**” открывает окно, показанное на рис. 1.11.

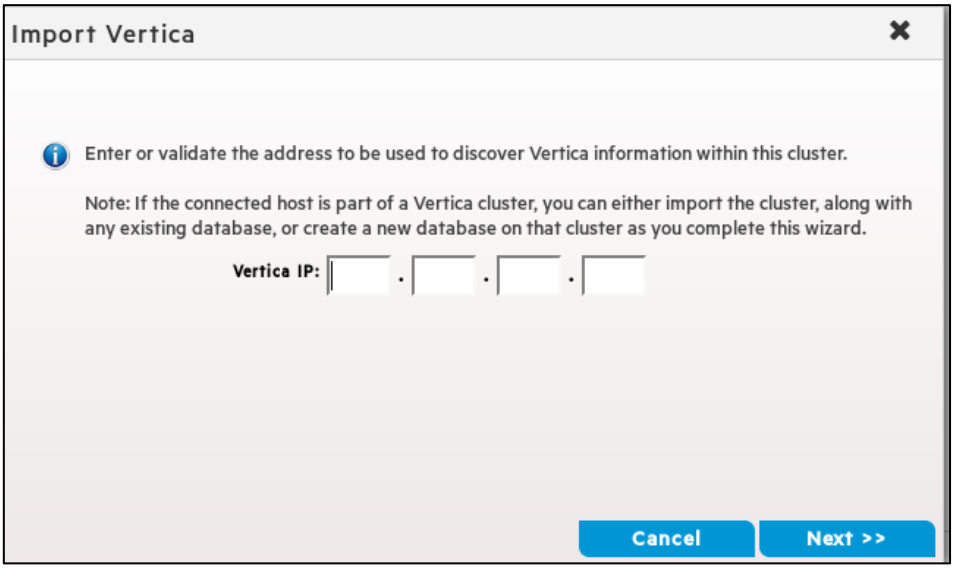


Рисунок 1.11 – Окно для ввода адреса импортируемого кластера

В этом окне необходимо задать IP адрес импортируемого кластера.

Пункт меню “**Additional Import Options**” в рассматриваемой версии системы открывает окно с одним пунктом меню, показанное на рис. 1.12.

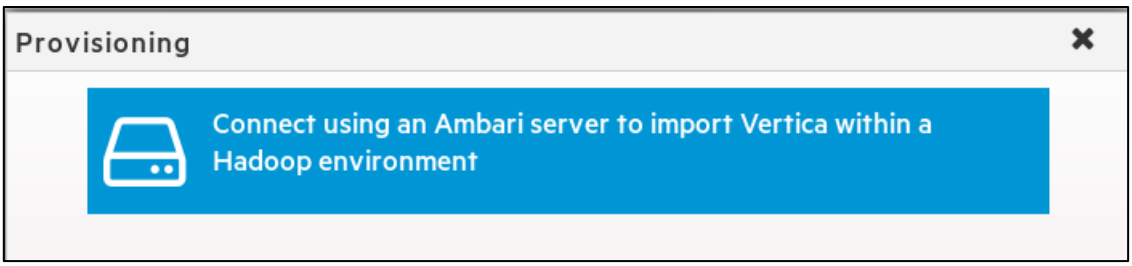


Рисунок 1.12 – Окно для импорта из системы Hadoop

1.4 Область “Manage”

Область содержит одну кнопку “**View Your Infrastructure**” (Просмотреть инфраструктуру). При нажатии на эту кнопку открывается окно, показанное на рис. 1.13.

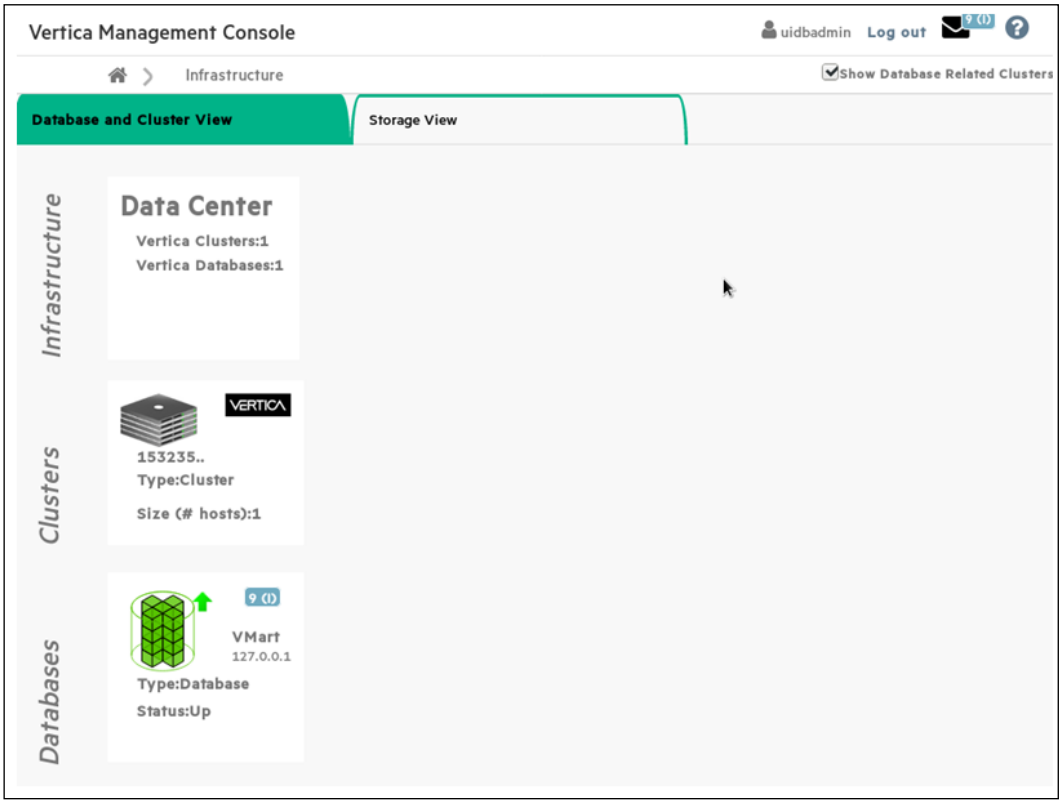


Рисунок 1.13 – Меню в режиме “Infrastructure”

Окно в свою очередь состоит из трех полей. Верхнее поле **Infrastructure** содержит информацию об инфраструктуре всей системы. На рис. 1.13 показано, что система состоит из одного кластера и одной базы данных.

При щелке мыши на поле кластеров **Clusters** открывается окно с информацией о кластере и подменю, показанное на рис. 1.14.

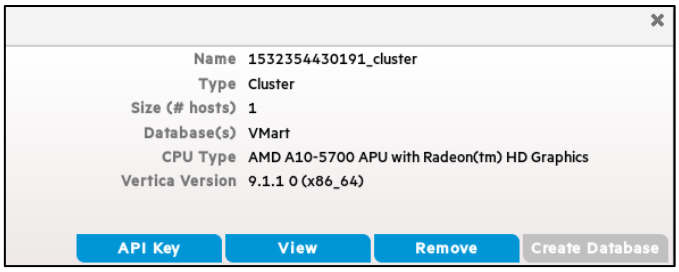


Рисунок 1.14 – Меню действий с кластером

В меню имеется четыре кнопки. Последняя кнопка “**Create Database**” недоступна, так как для этого действия надо закрыть активную базу данных, что можно выполнить из другого места.

По кнопке “**API Key**” – открывается окно для ввода ключа для работы с кластером, как показано на рис. 1.15.

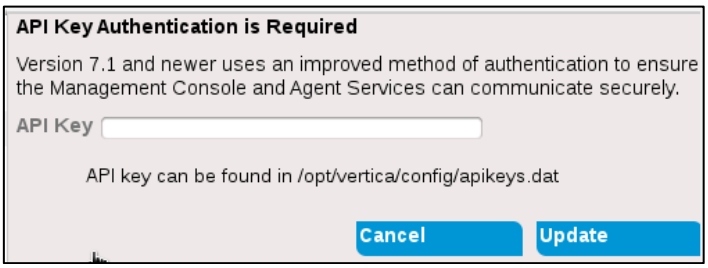


Рисунок 1.15 – Ввод ключа

По кнопке “**View**” – открывается окно с информацией о кластере, как показано на рис. 1.16.

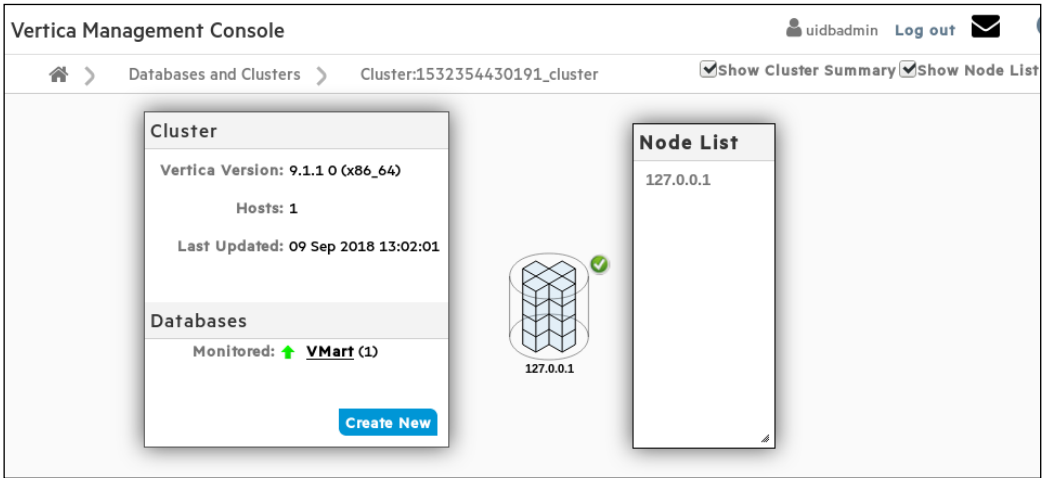


Рисунок 1.16 – Информация о кластере

В окне показываются существующие базы данных с их IP адресами и

два дополнительных поля, которые можно убрать, если изменить маркеры в пунктах в правом верхнем углу окна.

В поле “**Cluster**” приводятся данные о кластере и его базах данных. Здесь имеется кнопка для создания новой базы данных.

В поле “**Node List**” приводится список узлов большой базы данных.

При нажатии на кнопку “**Create New**” открывается окно для создания новой базы данных в кластере, показанное на рис. 1.17.

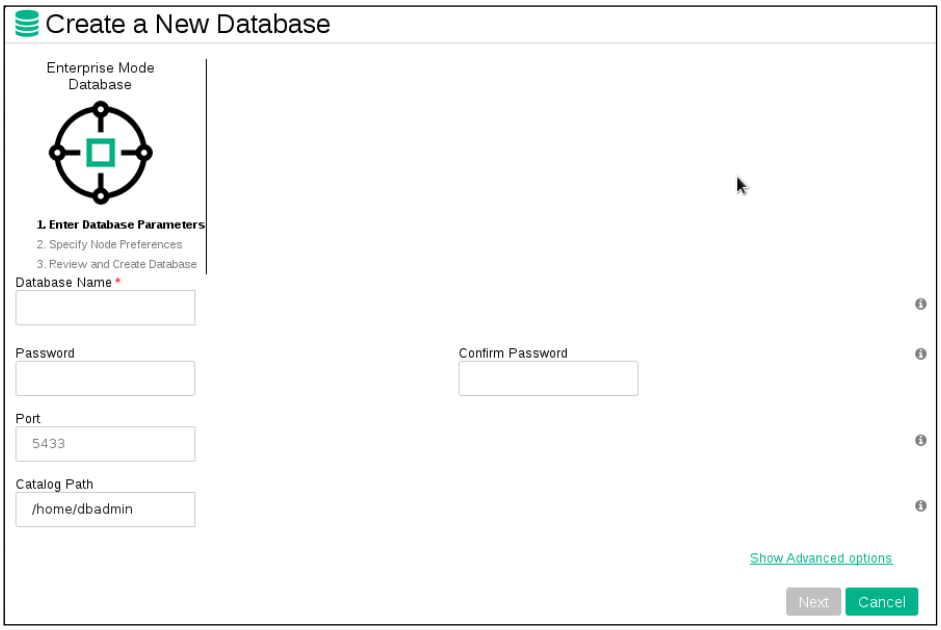


Рисунок 1.17 – Окно для создания новой базы данных

Вопросы создания новой базы данных будут рассматриваться в следующем практикуме.

По кнопке “**Remove**” – удаляется кластер. Кнопка опасная, в связи с чем при ее нажатии появляется предупреждение, показанное на рис. 1.18.

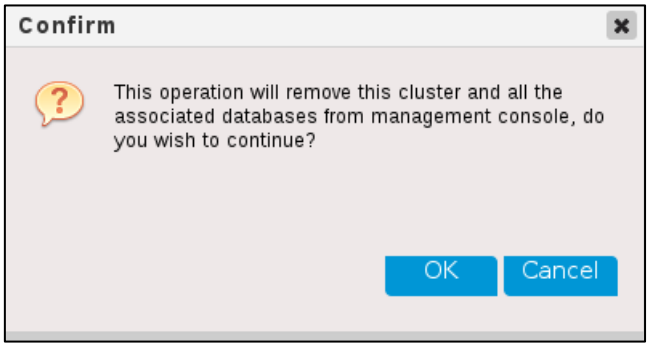


Рисунок 18 – Предупреждение при попытке удалить кластер

Это действие не выполнять ни в коем случае. Иначе придется переустанавливать систему.

При нажатии на поле **Databases** открывается окно с информацией о

базе данных и подменю, показанное на рис. 1.19. Открытые базы данных имеют зеленый цвет. Закрытые – красный цвет.

Для того чтоб остановить базу данных нужно нажать кнопку “**Stop**”.

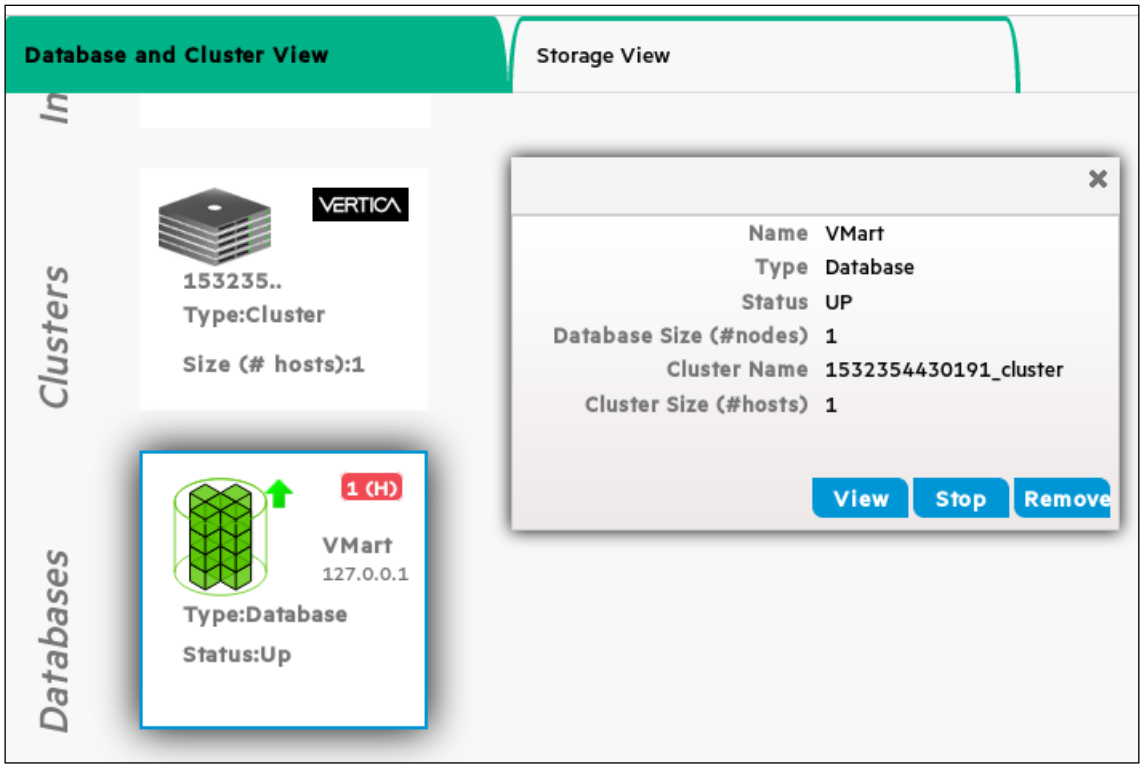


Рисунок 1.19 – Меню управления базой данных

Упражнение 1.1. Остановить работу базы данных “VMart”, нажав кнопку “**Stop**”. Должно появиться предупреждение, показанное на рис. 1.20.

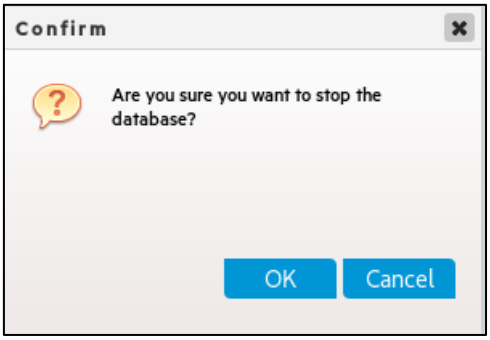


Рисунок 1.20 – Предупреждение при остановке базы данных

Нажать кнопку “**OK**”. Иконка базы данных должна изменить цвет на красный, как показано на рис. 1.21.

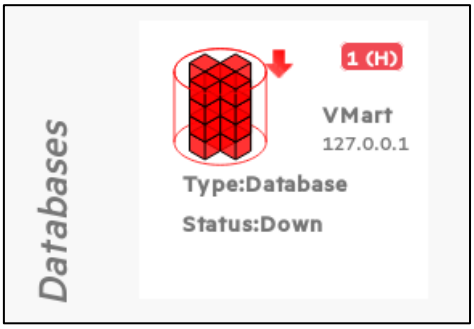


Рисунок 1.21 – Изображение остановленной базы данных

Если теперь нажать левой кнопкой мыши на красном изображении базы данных, то открывается окно с подменю, показанное на рис. 1.22.

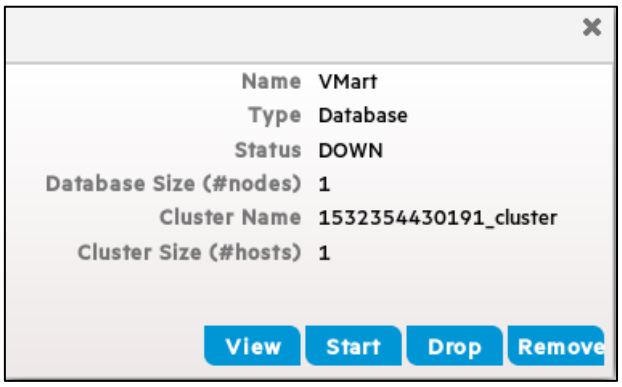


Рисунок 1.22 – Окно для действий с закрытой базой данных

Кнопка “**Start**” служит для запуска базы данных. Сейчас статус базы данных “DOWN”, то есть база данных выключена. Появилась дополнительная кнопка “**Drop**” (Удалить). Закрытую базу данных можно полностью удалить из системы. При этом появляется окно предупреждения, показанное на рис. 1.23.

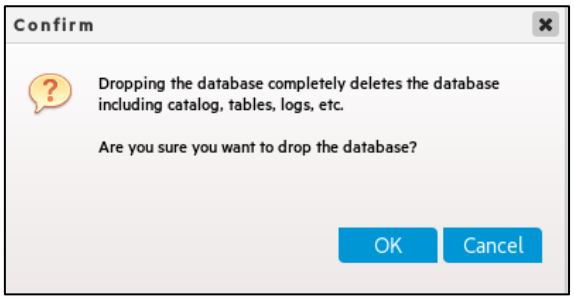


Рисунок 1.23 – Предупреждение при удалении базы данных

Демонстрационную базу данных не удалять!

Кнопка “**Remove**” удаляет базу данных из кластера, но не из системы. Предупреждения в этом случае нет. Восстановить базу данных можно будет

из окна с информацией о кластере с помощью кнопки “**Import Discovered**” (Импорт исключенного), как показано на рис. 1.24.

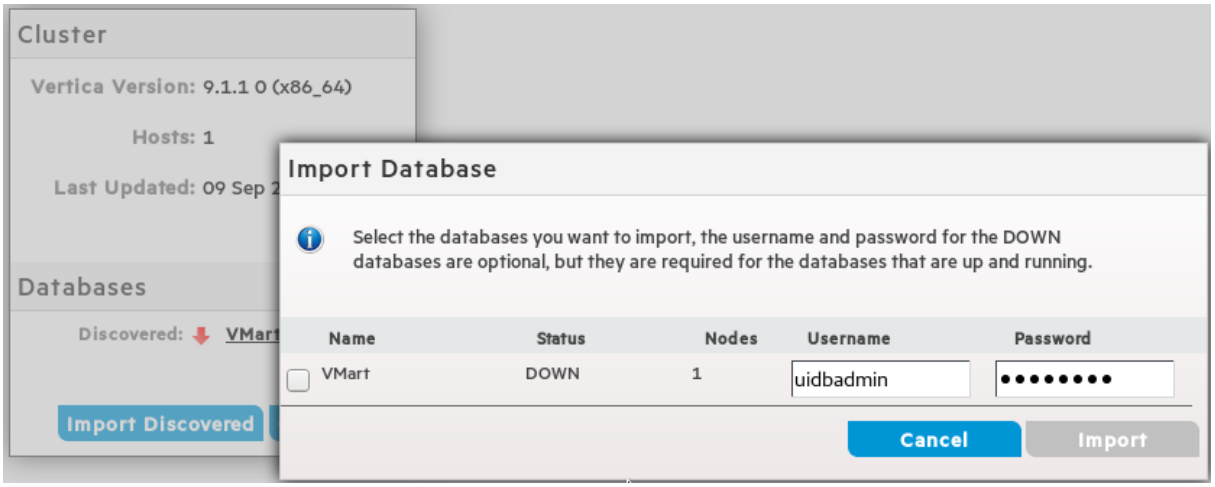


Рисунок 1.24 – Восстановление удаленной из кластера базы данных

При удачном восстановлении базы данных появляется сообщение, показанное на рис. 1.25.

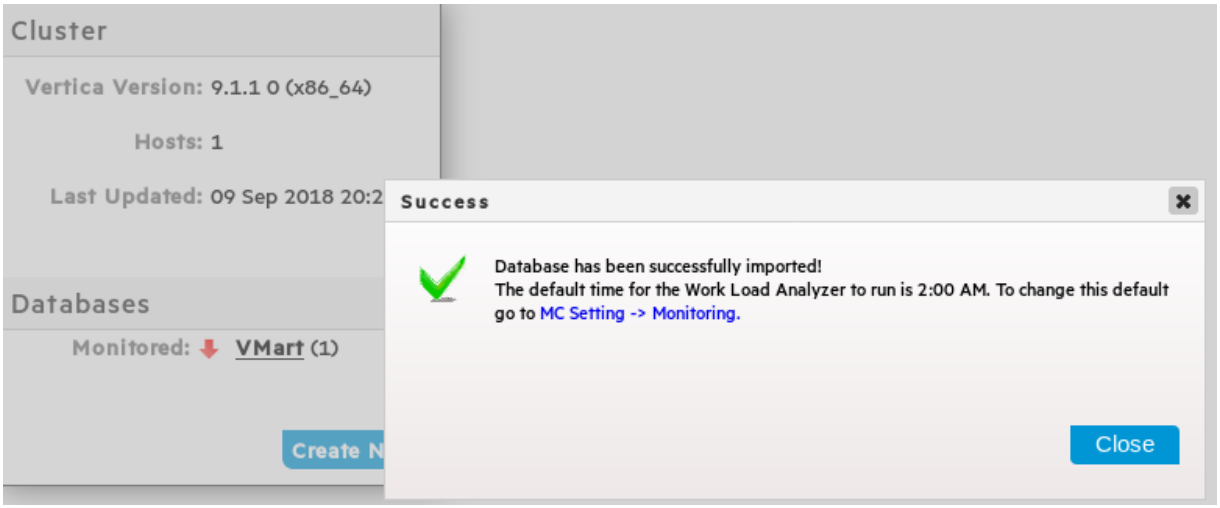


Рисунок 1.25 – Сообщение об удачном восстановлении базы данных

При повторном использовании базы данных могут возникнуть трудности в связи с возможным изменением политики доступа к базе данных. В этом случае потребуется использовать средства управления доступом к базе данных, которые будут рассмотрены в следующих практикумах. В связи с этим удалять базы данных из кластера надо только в том случае, если они больше не понадобятся.

Упражнение 1.2. Вновь запустить базу данных “VMart” с помощью кнопки “**Start**”. При этом появится предупреждение, показанное на рис. 1.26.

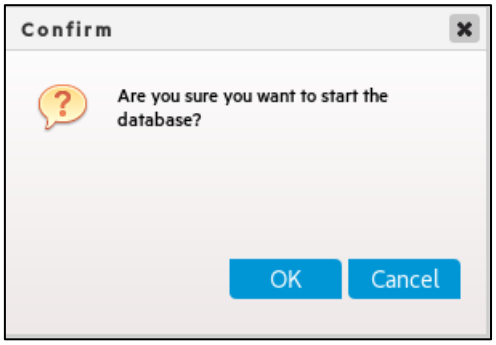


Рисунок 1.26 – Предупреждение при старте базы данных

Цвет иконки должен поменяться на зеленый.

Самая интересная и важная кнопка “**View**”. При нажатии на эту кнопку открывается окно, показанное на рис. 1.27.

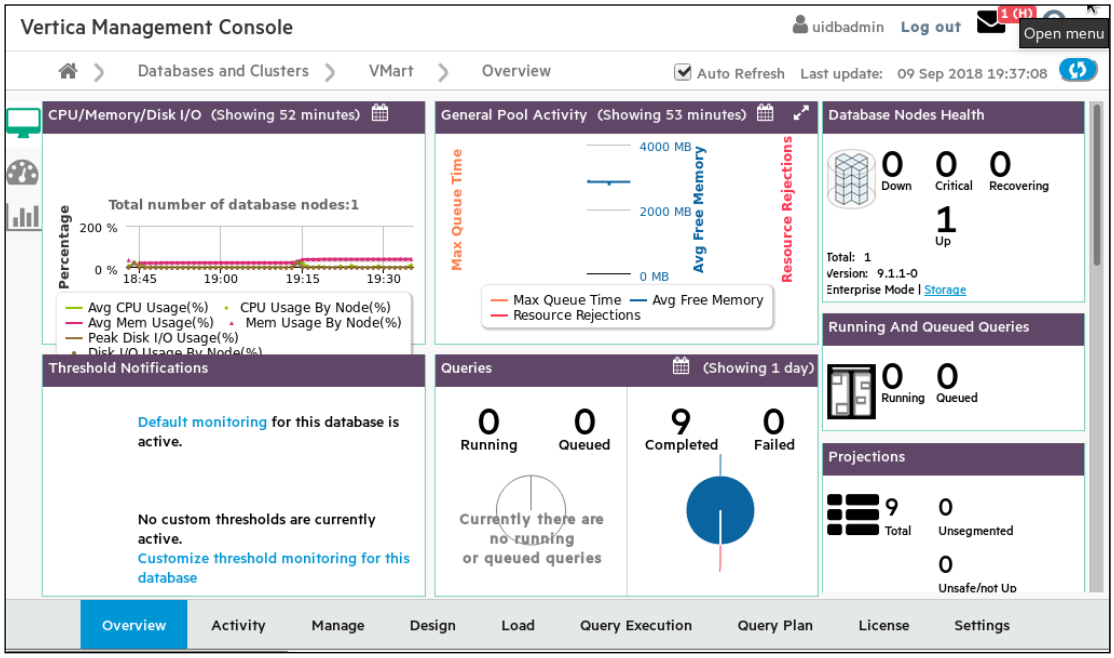


Рисунок 1.27 – Окно с информацией о базе данных

Окно содержит статистику использования базы данных и меню действий с базой данных. Это окно будет активно использоваться в течении всего практикума. Меню окна отражается снизу. В верхнем поле показан путь к окну: **Databases and clusters** > **VMart** > **Overview**. В данном практикуме для просмотра структуры базы данных воспользуемся пунктом меню рассматриваемого окна “**Activity**” (Действие). При выборе этого пункта меню открывается окно, показанное на рис. 1.28.

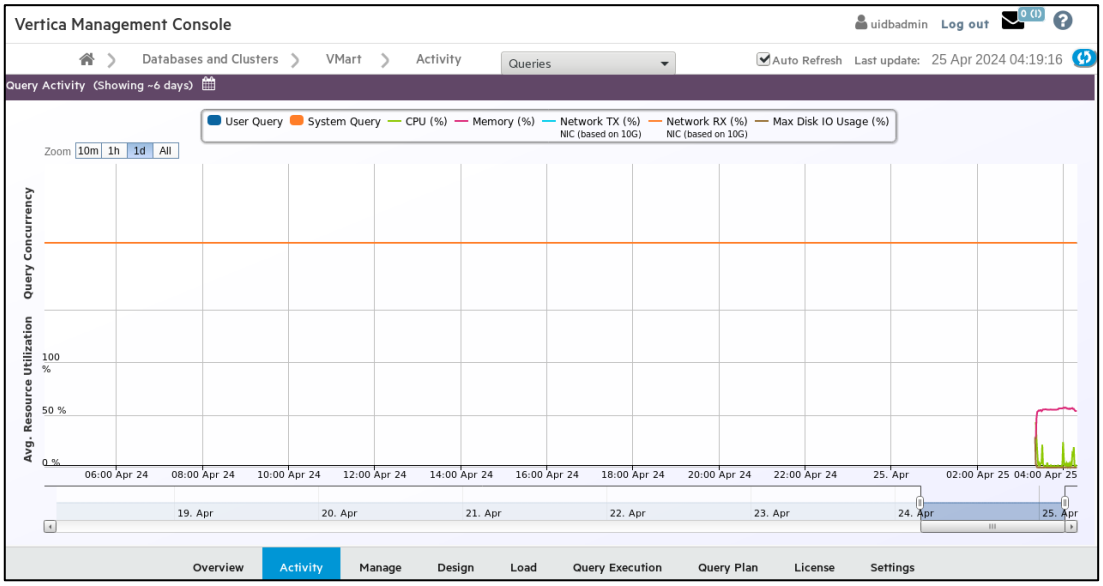


Рисунок 1.28 – Окно статистики по действиям с базой данных

В верхней части окна имеется открывающийся список **Queries** (Запросы) со стрелкой. При нажатии на эту стрелку открывается дополнительный список видов информации, показанный на рис. 1.29.

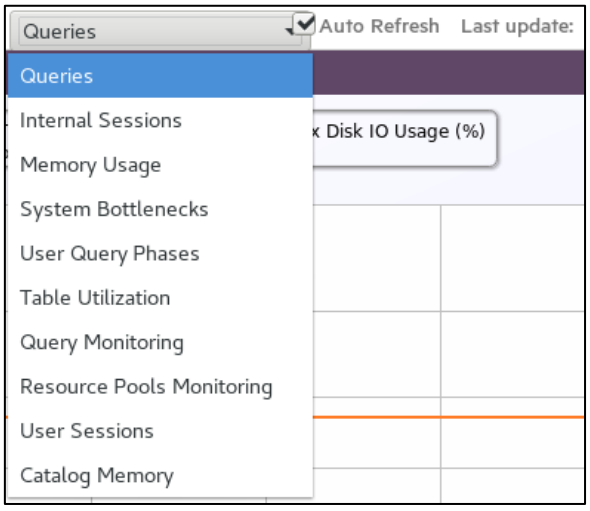


Рисунок 1.29 – Меню с видами информации о базе данных

Пункты данного списка будут изучаться в процессе всего практикума. В данном практикуме рассмотрим пункт меню “**Table Utilization**” (Использование таблиц). При выборе этого пункта откроется окно, показанное на рис. 1.30.

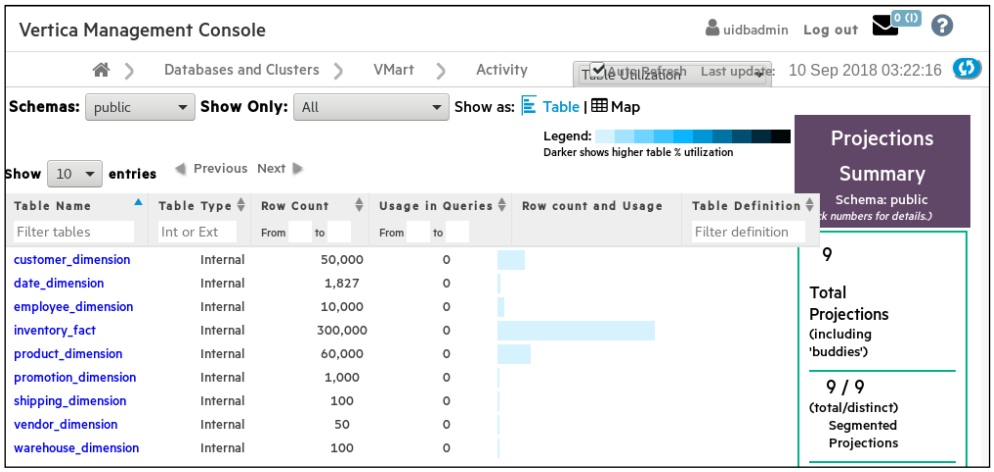


Рисунок 1.30 – Отношения базы данных

В теории баз данных таблицы называются отношениями.

База данных делится на схемы. Для выбора схемы надо открыть список схем, как показано на рис. 1.31.

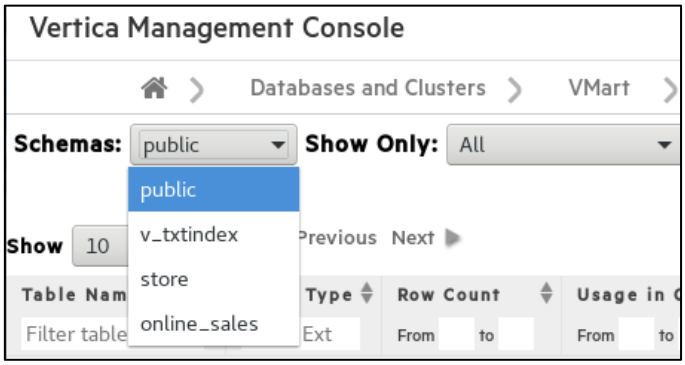


Рисунок 1.31 – Список схем базы данных

Окно на рис. 1.30 показывает список всех отношений схемы “public” демонстрационной базы данных VMart. Общая схема базы данных показана в приложении. При наведении курсора на конкретное отношение появляется всплывающее окно со статистической информацией об использовании этого отношения, как показано на рис. 1.32.

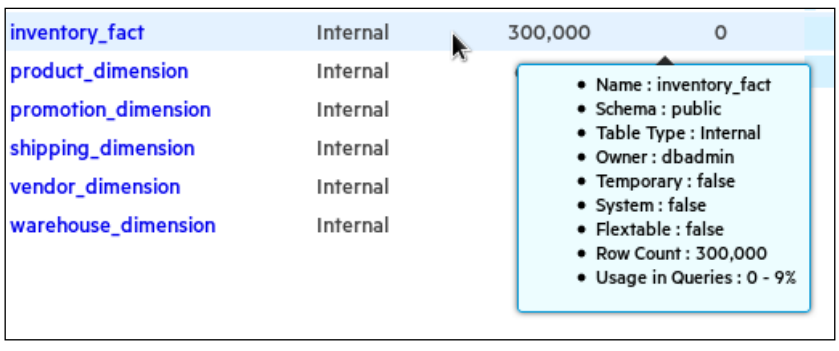


Рисунок 1.32 – Окно со статистикой отношения “inventory_fact”

При нажатии мышью на имени конкретного отношения открывается окно со структурой отношения, как показано на рис. 1.33. При этом показываются имена всех атрибутов отношения, их форматы и ряд другой информации, которая будет рассматриваться в дальнейшем.

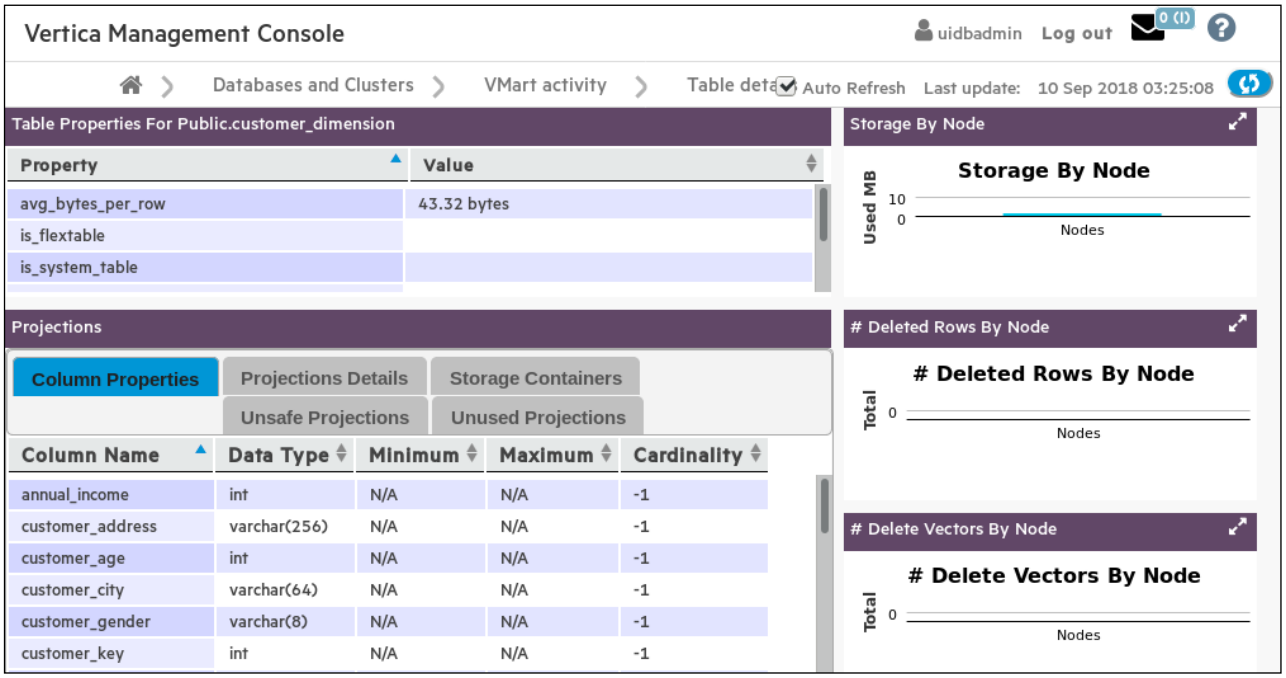


Рисунок 1.33 – Структура отношения “custome_dimension”

Упражнение 1.3. Просмотреть структуру базы данных примера для всех схем.

Окно инфраструктуры имеет еще одну закладку “**Storage View**” (Обзор хранилища данных), пример содержимого которой показано на рис. 1.34.

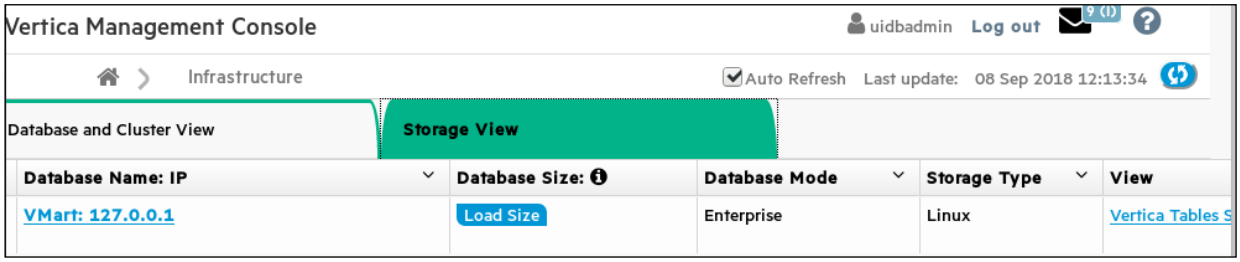


Рисунок 1.34 – Закладка с информацией о базах данных

При нажатии на активные тексты с голубым цветом открывается дополнительная информация о хранении данных: размер базы данных (Data base Size), тип хранилища (Storage Type), адрес данных в файловой системе (Vertica Tables Storage).

1.5 Область консоли «MC Tools»

Поле главного меню “**MC Tools**” (Инструменты консоли менеджера) содержит следующие три кнопки:

- кнопка “**MC Settings**” вызывает окно инструментов системы;
- кнопка “**Message Center**” вызывает окно центра сообщений;
- кнопка “**MC Diagnostic**” вызывает окно диагностики консоли менеджера.

При нажатии на кнопку **MC Setting** открывается окно установок системы, показанное на рис. 1.35.

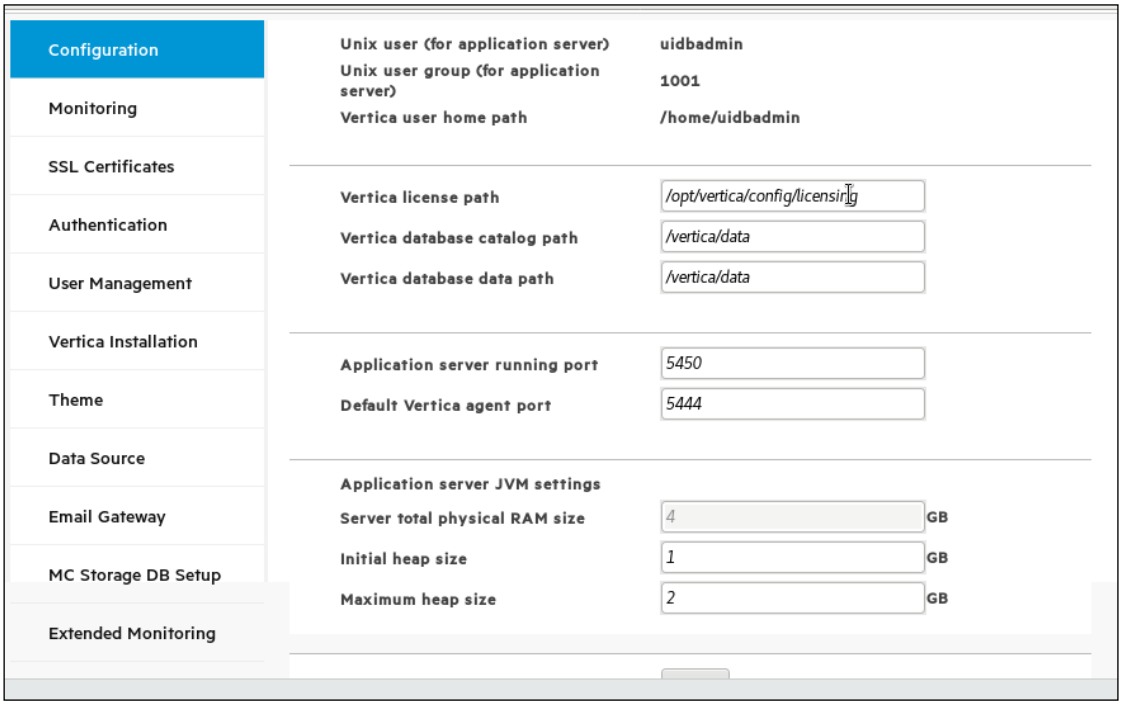


Рисунок 1.35 – Окно установок системы

Пункты данного режима будут рассматриваться по мере необходимости.

Кнопка “**Message Center**” вызывает окно центра сообщений, как показано на рис. 1.36.

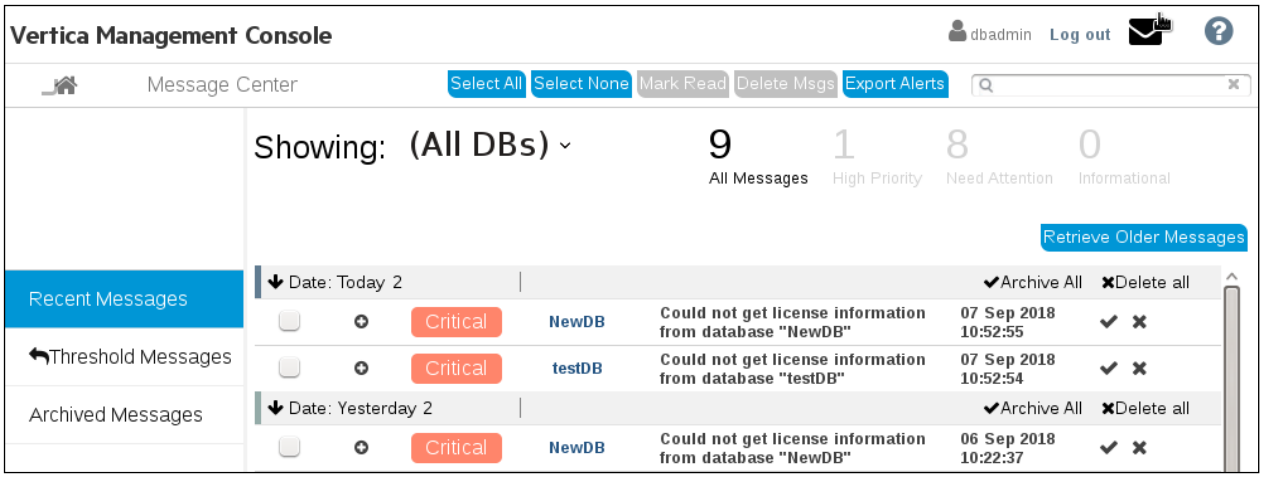


Рисунок 1.36 – Журнал сообщений

Можно просмотреть текущие сообщения (Recent Messages), нерассмотренные сообщения (Threshold Messages) и заархивированные сообщения (Archived Messages).

Кнопка “**MC Diagnostic**” вызывает окно диагностики консоли менеджера, показанное на рис. 1.37. Этот режим позволяет просмотреть результаты диагностики системы и дополнительную информацию.

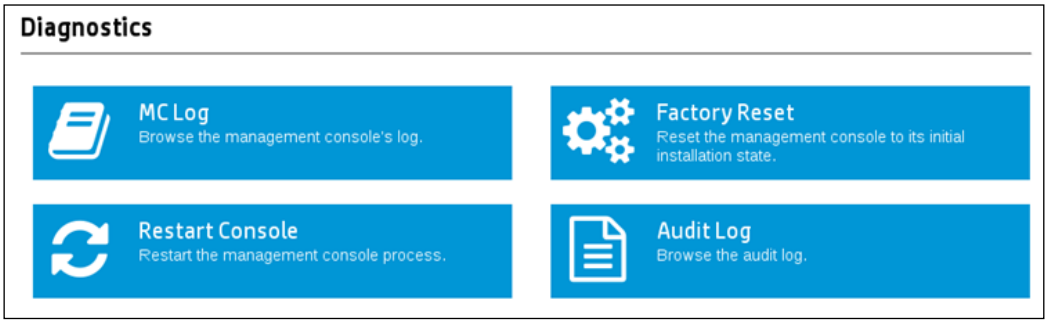


Рисунок 1.37 – Меню для режима “Diagnostics”

При выборе пункта меню “**MC Log**” (Журнал консоли менеджера) открывается окно, показанное на рис. 1.38.

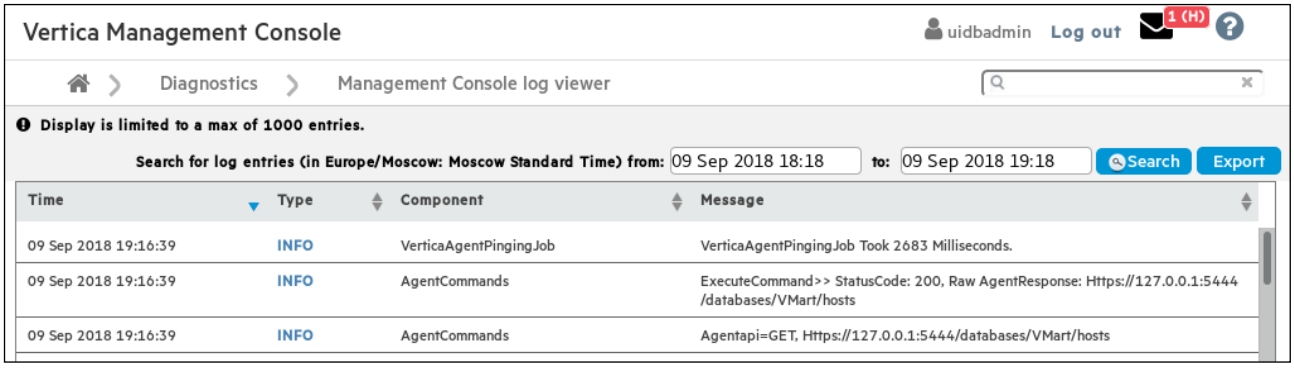


Рисунок 1.38 – Журнал консоли менеджмента

Окно содержит информацию о последних событиях в системе. Имеется возможность сохранить журнал событий (log). Для этого надо нажать кнопку “**Export**”. В результате откроется окно с радио кнопками, показанное на рис. 1.39.

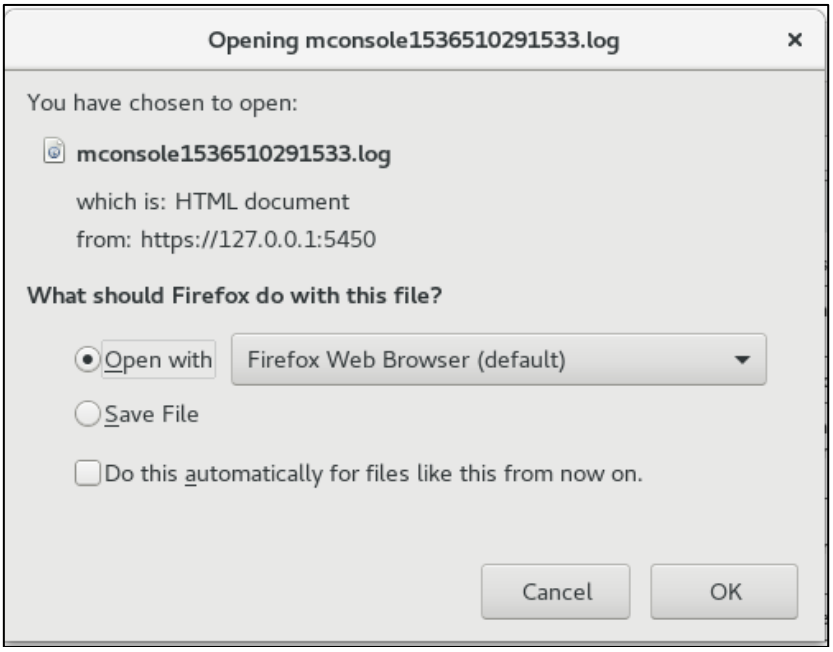


Рисунок 1.39 – Окно для сохранения журнала консоли во внешнем файле

Пункт меню диагностики “**Factory reset**” предназначен для сброса консоли менеджера базы данных в исходное состояние. При выборе этого пункта меню выводится на экран предупреждение, показанное на рис. 1.40.

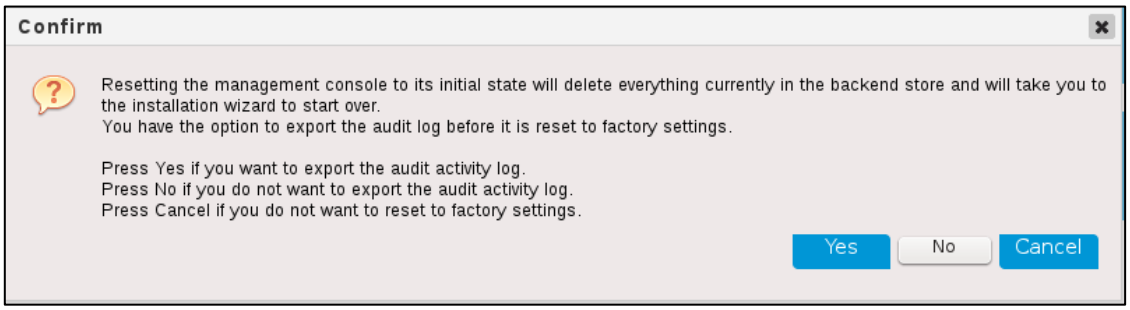


Рисунок 1.40 – Предупреждение при попытке сброса состояния консоли менеджера

Пункт меню “Restart console” позволяет вновь загрузить консоль менеджера. При этом появляется предупреждение, показанное на рис. 1.41, для подтверждения действия. При перезагрузке консоли вновь появится окно идентификации “Login page”.

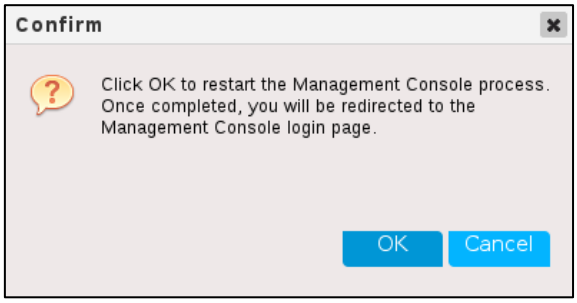


Рисунок 1.41 – Предупреждение при перезагрузке консоли

При выборе пункта меню диагностики “**Audit Log**” будет выведен на экран журнал событий, как показано на рис. 1.42.

Рисунок 1.42 – Журнал событий

1.6 Область «Recent Databases»

Область содержит информацию об имеющихся базах данных и их состоянии. При нажатии на иконку открытой базы данных открывается окно со статистикой базы данных, показанное ранее на рис. 1.27. В таблице баз данных имеется два столбца, показанные на рис. 1.43.

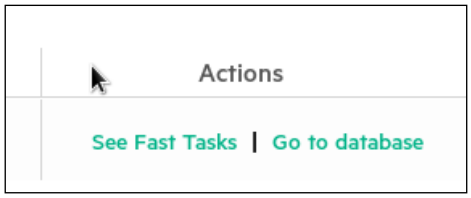


Рисунок 1.43 – Действия с базой данных

При нажатии на пункт “**See Fast Tasks**” открывается меню для быстрого доступа к основным действиям с базой данных, показанное на рис. 1.44.

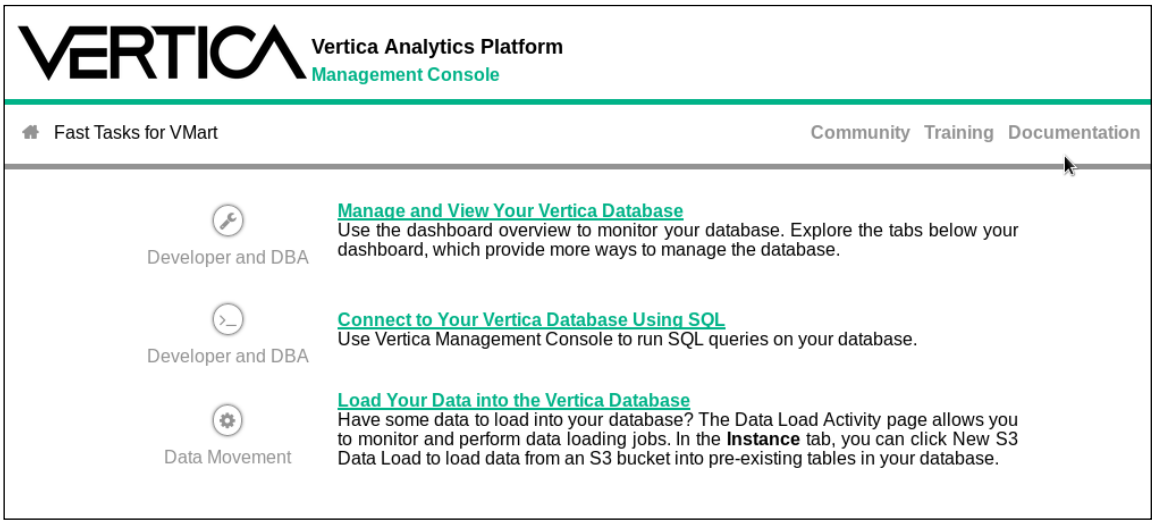


Рисунок 1.44 – Быстрые кнопки доступа к основным действиям

Доступ осуществляется к следующим трем возможным режимам консоли менеджера:

- “**Manage and View Vertica Database**” – управление и просмотр базы данных;
- “**Connect to Your Vertica Database Using SQL**” – взаимодействие с базой данных с помощью языка SQL;
- “**Load Your Data into the Vertica Database**” – загрузка данных в базу данных.

При использовании этого меню управление передается к уже рассмотренным пунктам интерфейса администратора базы данных.

1.7 Выход из системы

При окончании работы с системой первоначально надо выйти из консоли менеджера, воспользовавшись кнопкой “**Log out**” в верхнем правом углу окна, как показано на рис. 1.45.

Рисунок 1.45 – Кнопка выхода из консоли менеджера

Затем надо остановить виртуальную машину с помощью кнопки выключения машины, как показано на рис. 1.46.

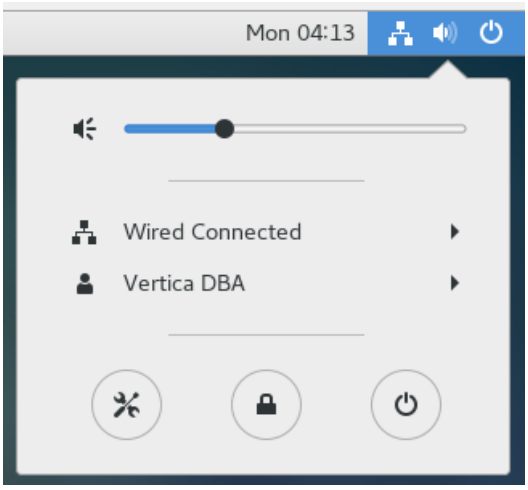


Рисунок 1.46 – Выключение виртуальной машины

1.8 Порядок выполнения работы

1. Загрузить виртуальную машину с системой.
2. Выполнить упражнения, описанные в данном практикуме.
3. Оформить отчет о проделанной работе.

1.9 Содержание отчета

Отчет должен содержать следующую информацию: копии окон экрана с выполненными упражнениями, выводы.

1.10 Контрольные вопросы

1. Отличие интерфейса Linux от Windows.
2. Что такое Big Data?
3. Что такое виртуальная машина?
4. Как войти в систему Vertica?
5. Что такое консоль менеджера?
6. Какие основные области консоли менеджера?
7. Как открыть базу данных?
8. Как закрыть базу данных?
9. Что такое отношение?
10. Что такое атрибуты отношения?
11. Что такое схема базы данных?
12. Как просмотреть список отношений, входящих в схему?
13. Как просмотреть список атрибутов отношения?

Глава 2. Создание базы данных в системе Vertica

2.1 Кластеры системы Vertica

Vertica – это система управления распределенной базой данных, которая работает на базе параллельной обработки разнообразных массивов информации. Система поддерживает стандартный язык манипулирования данными SQL плюс множество методов бизнес-аналитики. Система поддерживает кластерную архитектуру. На рис. 2.1. показана структура кластера системы. Кластер системы состоит из компьютеров, которые логически объединены в узлы.

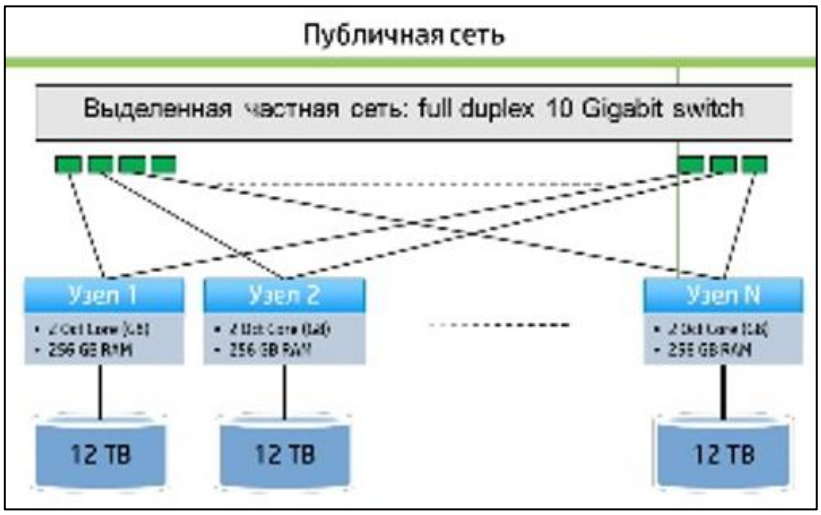


Рисунок 2.1 – Схема кластера системы Vertica [13]

Использование распределенных кластерных технологий обработки данных позволяет значительно увеличить производительность системы, снизить общий объем хранимых данных и сократить время поиска информации [10]. Столбиковая технология хранения данных дает возможность избирательного взаимодействия с данными, используя для выполнения запросов не всю базу данных, а только нужные атрибуты, необходимые в запросе.

Также удается достичь ускорения обработки большого количества параллельных запросов от нескольких пользователей за счет использования различных методов индексации и сортировки информации в разных копиях столбцов и в разных проекциях, которые выбираются автоматически.

Используемая система сжатия данных позволяет хранить множество копий одних и тех же колонок в разных «проекциях» базы данных, которые представляют собой выборки столбцов по различным критериям. При этом возможно хранение не только различных копий элементов базы данных на разных дисках, но и разделение «проекций» по значениям полей на сегменты, которые могут находиться и обрабатываться на разных компьютерах.

В тоже время внешне база данных представляется для пользователей в виде классической реляционной модели, в которой все данные

представляются в виде таблиц/отношений. Реляционная модель обладает многими полезными свойствами и обеспечивает единообразие представления данных. Все объекты модели представляются в концептуальной модели данных совершенно одинаково - таблицами. При этом, конечно, усложняется процесс манипулирования данными.

Система содержит реляционно-полный язык манипулирования данными SQL. Это означает, что язык манипулирования данными должен выполнять любую операцию реляционной алгебры или реляционного исчисления. Более того, используемый язык описывает любой поисковый запрос в виде операций с таблицами, а не с кортежами.

Для формального описания таблицы используется теоретико-множественное понятие отношения. Список названий столбцов таблицы (атрибутов) именуют схемой отношения и обозначают R ($A1$, $A2$, …, An).

Совокупность схем отношений, используемых для представления концептуальной модели, называется схемой реляционной базы данных, а текущие значения соответствующих отношений – реляционной базой данных. В качестве основного недостатка реляционной модели можно указать дублирование информации при представлении связей.

2.2 Создание новой базы данных в системе Vertica

Будем предполагать, что виртуальная машина с системой Vertica уже запущена. Будем работать в режиме консоли менеджера, как было рассмотрено в предыдущем лабораторном практикуме. На рис. 2.2 показано первоначальное окно консоли менеджера базы данных, в котором надовы брать управление в поле «**Manage**».

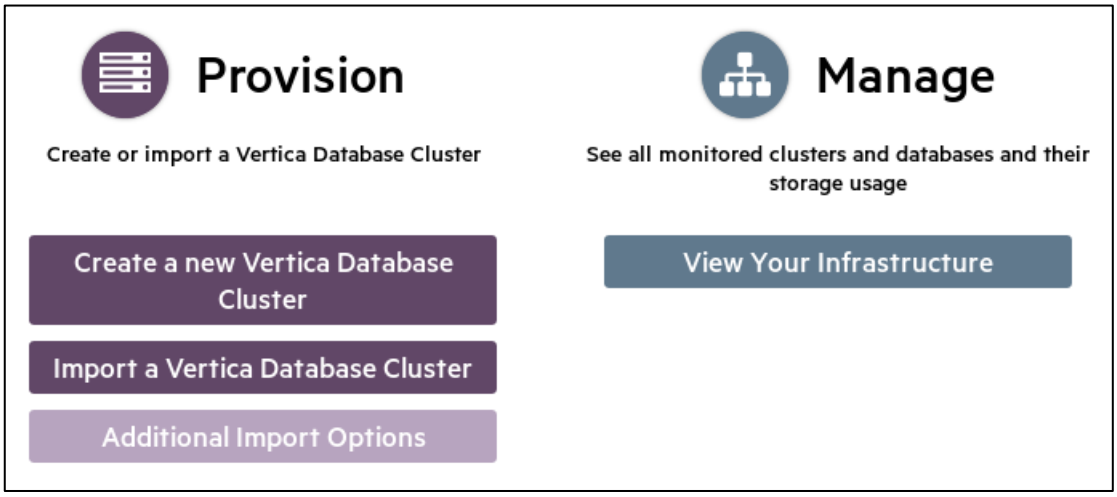


Рисунок 2.2 – Главное меню консоли менеджера

В поле надо выбрать пункт меню “**View Your Infrastructure**” (Обзор Вашей инфраструктуры). В результате будет видна структура системы, как показано на рис. 2.3.

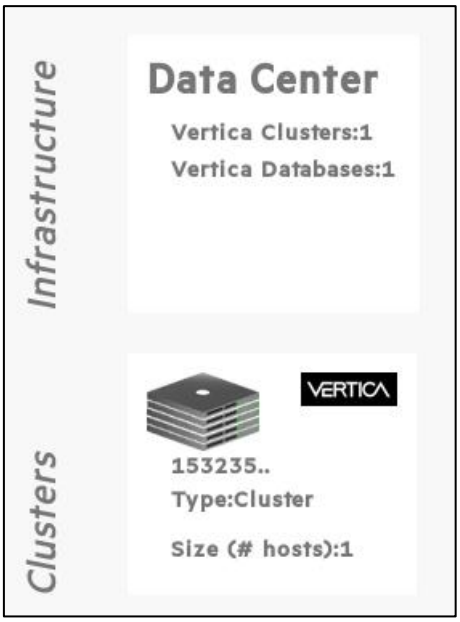


Рисунок 2.3 – Структура системы

При щелчке мышью на иконке кластеров, открывается окно, показанное на рис. 2.4.

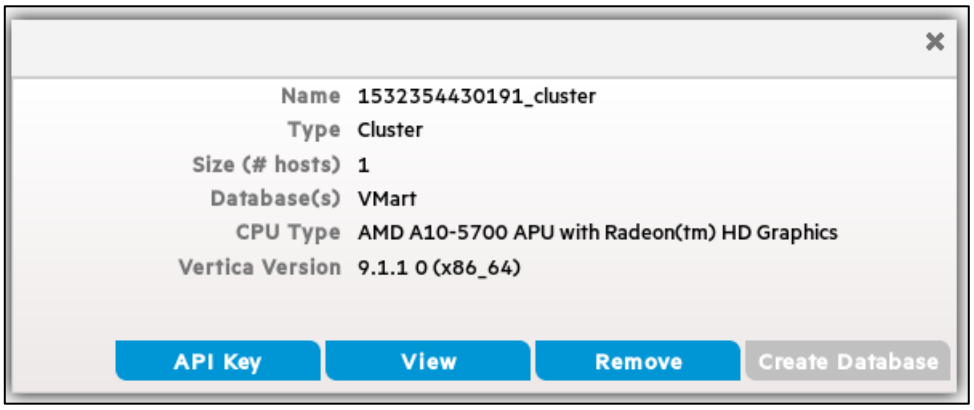


Рисунок 2.4 – Выбор пункта меню обзора кластера

Следует обратить внимание, что пункт меню «**Create Database**» в окне неактивен. Для просмотра информации о кластере в этом окне надо выбрать пункт “**View**” (Обзор). Откроется окно, показанное на рис. 2.5.

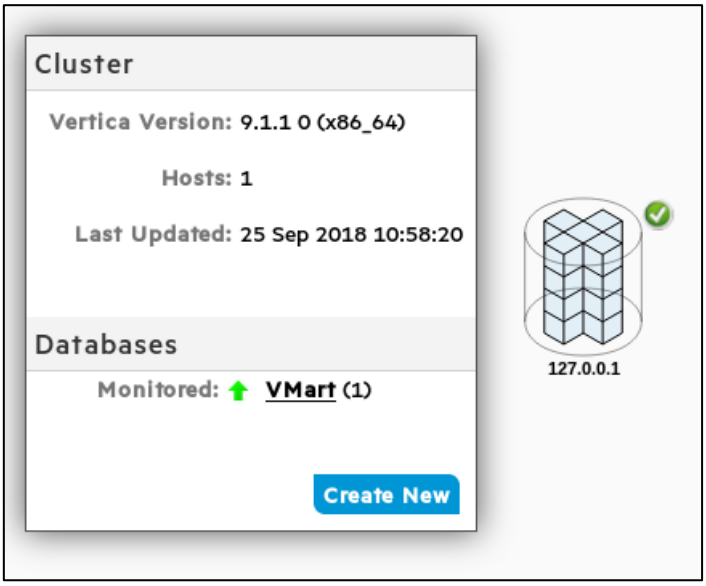


Рисунок 2.5 – Пункт меню создания новой базы данных в кластере

В информационном окне имеется кнопка “**Create New**” (Создать новую базу данных). При нажатии на эту кнопку будет выдано сообщение, показанное на рис. 2.6.

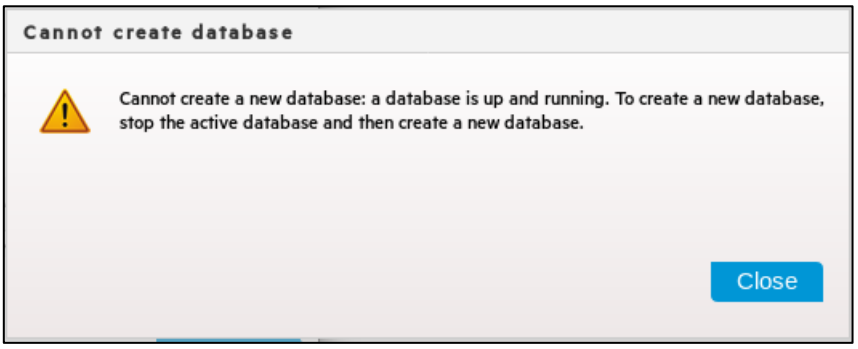


Рисунок 2.6 – Сообщение о том, что имеются открытые базы данных

В используемой версии системы не предусмотрено одновременное открытие более одной базы данных. Поэтому необходимо остановить работающую базу данных VMart. Зеленый цвет иконки показывает, что база данных открыта. При нажатии на иконку открывается окно, показанное на рис. 2.7.

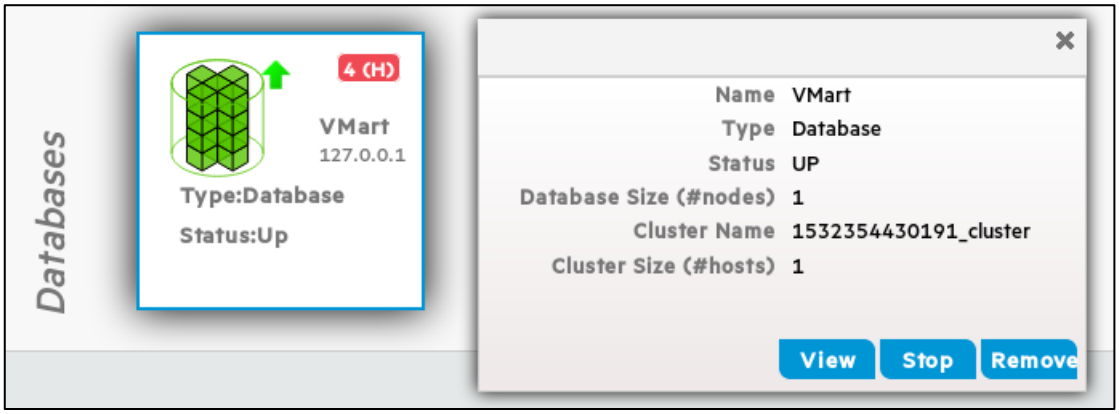


Рисунок 2.7 – Меню для остановки базы данных

Для остановки базы данных надо выбрать пункт меню “**Stop**”. Система выдаст предупреждение, показанное на рис. 2.8.

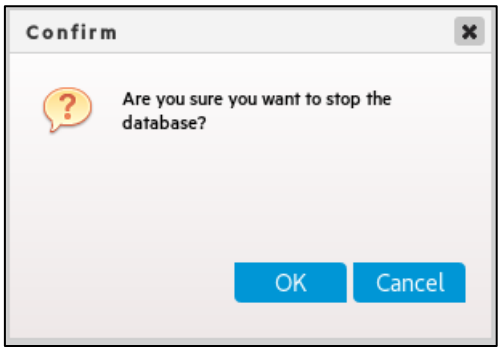


Рисунок 2.8 – Предупреждение перед закрытием базы данных

Если действие подтверждено (Кнопка **ОК**), то иконка базы данных меняет свой цвет на красный, как показано на рис. 2.9.

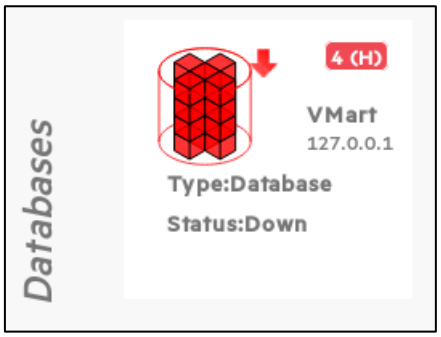


Рисунок 2.9 – База VMart закрыта

Теперь можно создавать новую базу данных с помощью окон, показанных на рис. 2.4 и 2.5. В этом случае при нажатии на кнопку «**Create New**» на рис. 2.5 открывается окно, показанное на рис. 2.10. Аналогичное окно появится при нажатии кнопки «**Create Data Base**» на рис. 2.4.

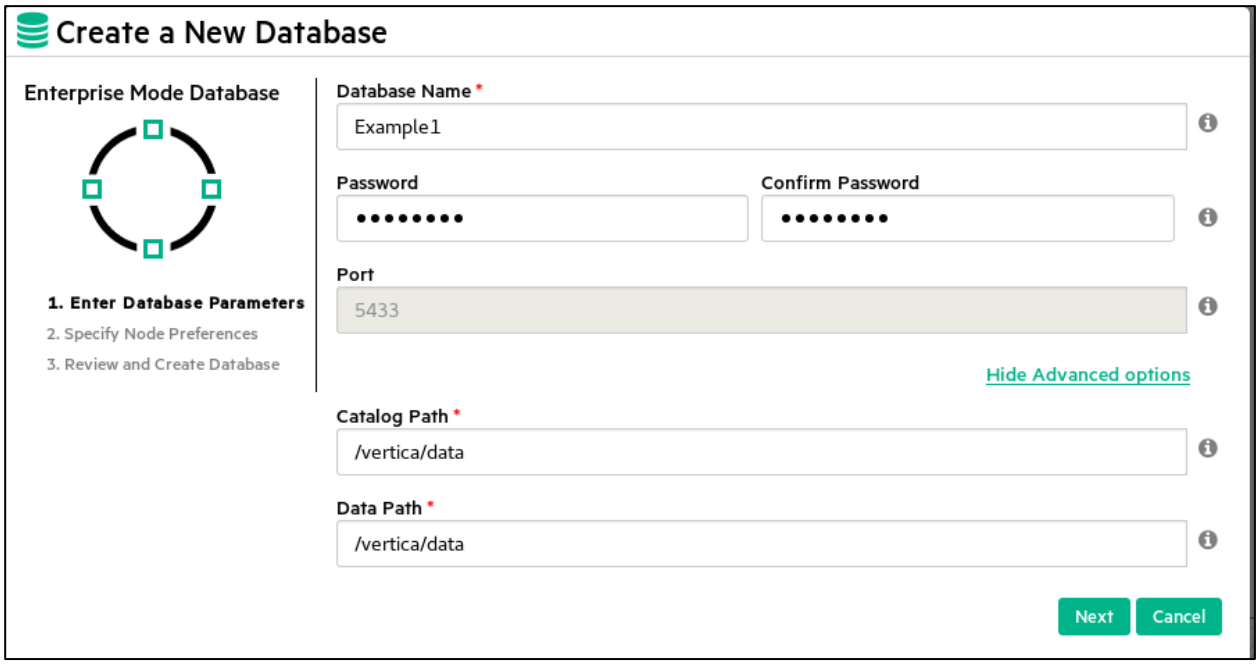


Рисунок 2.10 – Окно для задания парамтеров новой базы данных

В открывшемся окне надо задать имя новой базы данных и пароль для доступа к новой базе данных. Для рассмтариваемого примера имя базы данных “Example1”. Имя базы данных в системе Vertica должно сотоять из латинских букв и цифр. Для учебных целей можно оставить пароль “password”. Дополнительные параметры все можно оставить без изменений. После заполнения полей формы надо нажать кнопку «**Next**» (следующий).

На рис. 2.11 показано окно на следующем шаге задания параметров базы данных, а именно установление связи с узлом интранета. Этот пункт тоже надо оставить без изменений.

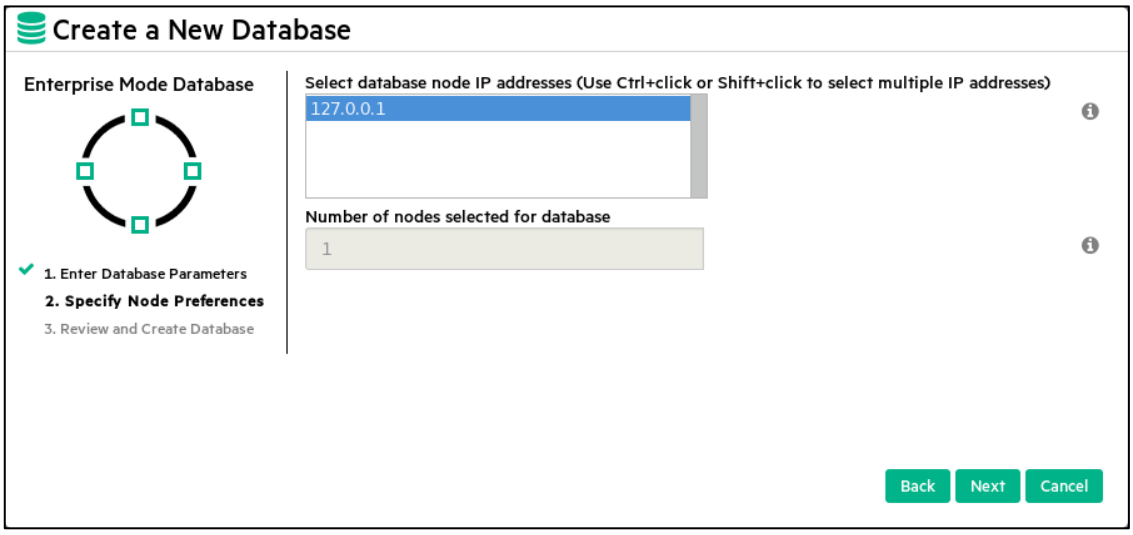


Рисунок 2.11 – Задание параметров узла интранет для базы данных

На последнем шаге создания базы данных дается обзор заданных параметров, как показано на рис. 2.12.

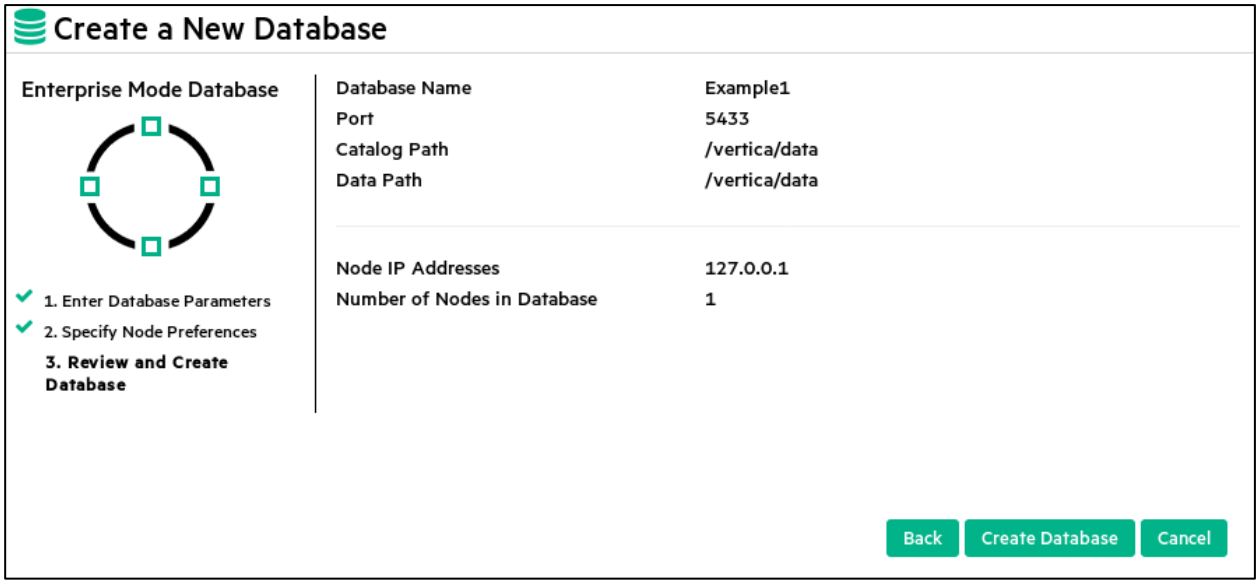


Рисунок 2.12 – Обзор параметров создаваемой новой базы данных

При нажатии кнопки “**Create Database**” начинается процесс создания базы данных с отражением шагов процесса, как показано на рис. 2.13.

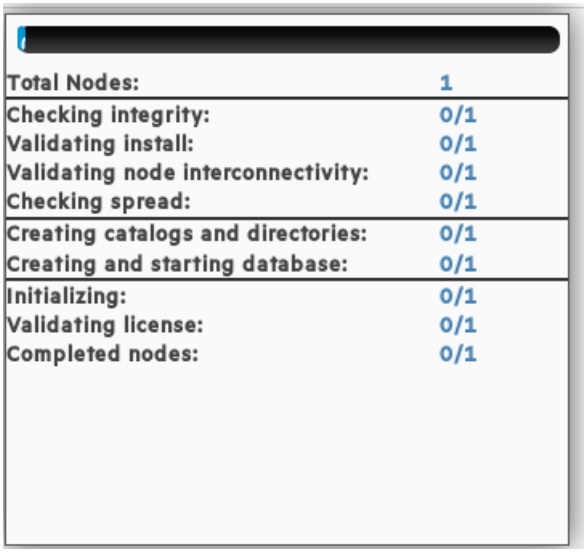


Рисунок 2.13 – Этапы процесса создания базы данных

При успешном выполнении этапов создания базы данных нужные пункты отмечаются зелеными галочками, как показано на рис. 2.14.

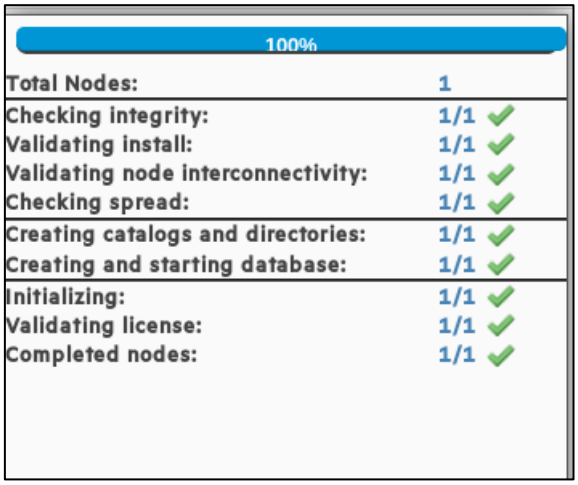


Рисунок 2.14 – Успешное выполнение этапов создания базы данных

При окончании создания базы данных должно появиться сообщение, показанное на рис. 2.15. Это требует определенного времени. В течении создания базы данных крутится кружок с сообщением Creating Data base. Все зеленные стрелочки и наличие картинки сновой базой данных недосточны для окончания процесса.

Внимание! Обязательно дождаться этого сообщения!

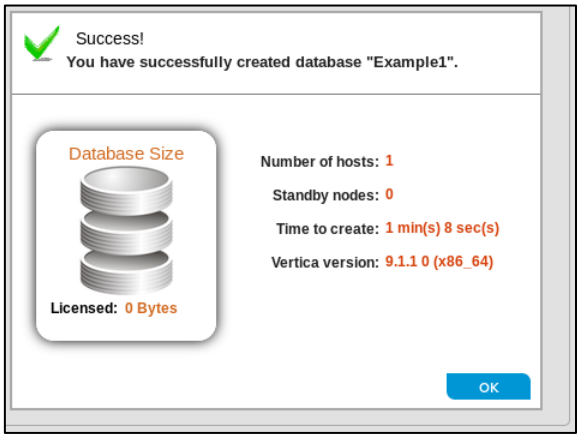


Рисунок 2.15 – Сообщение о создании новой базы данных

После нажатия кнопки ОК появится окно, показанное на рис. 2.16 с промежуточным состоянием базы данных, которое отмеченное желтым цветом.

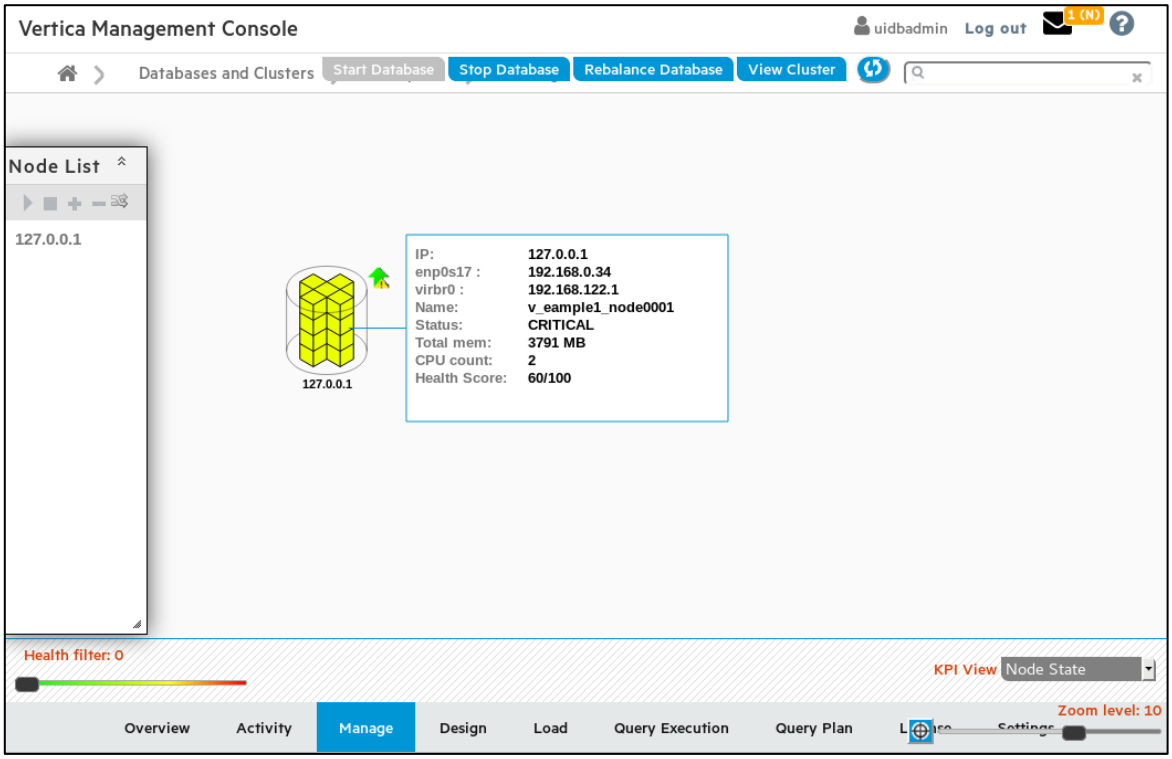


Рисунок 2.16 – Информация о промежуточном состоянии базы данных

При нажатии на кнопку «**Refresh the grid**» в верхнем меню окна, показанную на рис. 2.17, статус базы данных с предварительного состояния «CRITICAL» меняется на открытое состояние «UP».

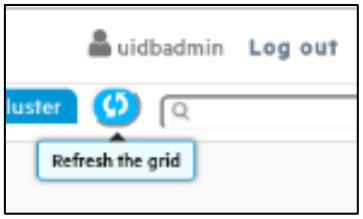


Рисунок 2.17 – Кнопка рестарта окна

Чтобы вернуться в окно «**Databases and Clusters**» надо нажать на соответствующую кнопку в верхнем меню окна, показанного на рис. 2.16.

Теперь в системе имеется две базы данных и на рис. 2.18 показана в всплывающем окне информация об открытой новой базе данных.

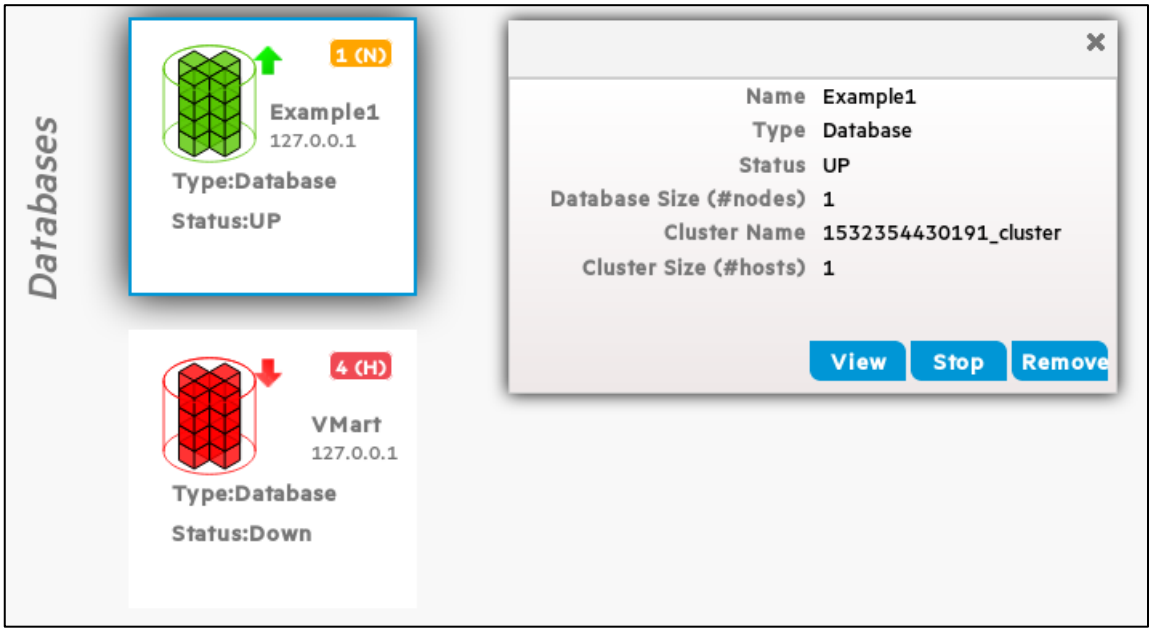


Рисунок 2.18 – Открытая новая база данных

Закрытая демонстрационная база данных VMart показана красным цветом и новая активная база данных Example1 зеленым цветом.

При новом входе в систему восстанавливается активной демонстрационная база данных, как показано на рис. 2.19.

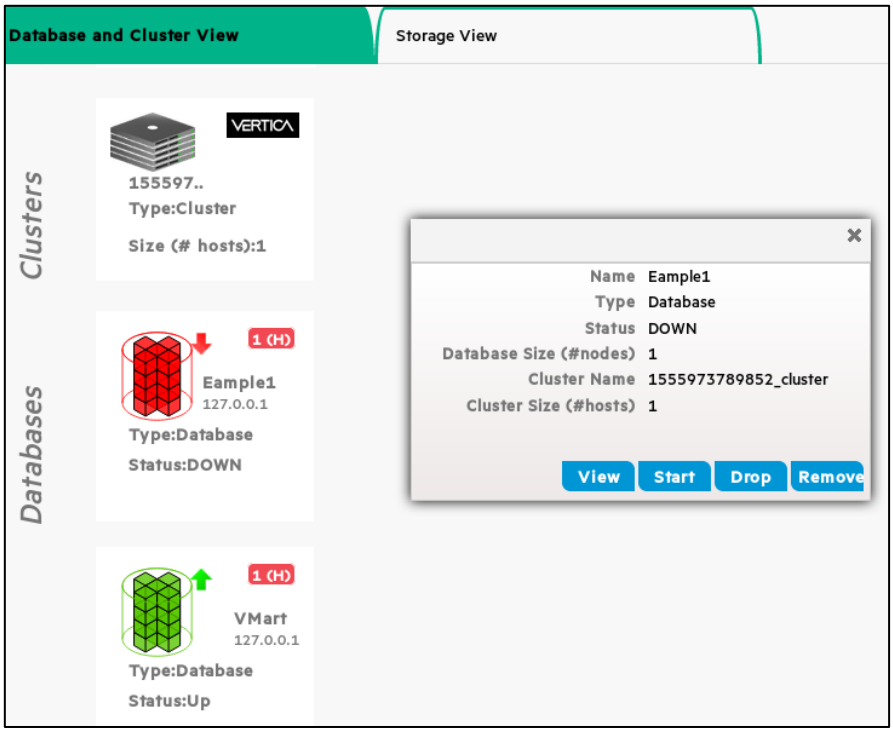


Рисунок 2.19 – Активность баз данных при новом входе в систему

Меню закрытой базы данных содержит пункт «**Start**» для перевода базы данных в активный режим и пункт «**Drop**» для удаления базы данных. В

демонстрационной версии системы для старта второй базы данных надо остановить другую активную базу данных.

При удалении базы данных система выдает сообщение с предупреждением о последствиях действия, как показано на рис. 2.20.

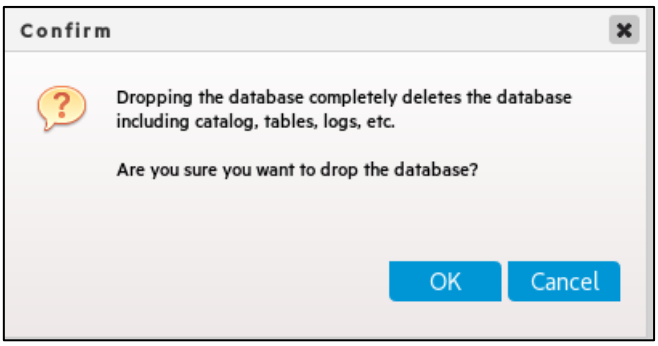


Рисунок 2.20 – Предупрждение при удалении базы данных

При удалении базы данных полностью удаляются все объекты, связанные с базой данных, такие как каталоги, таблицы, протоколы и другие.

Упражнение 2.1. Требуется создать новую базу данных со своим индивидуальным именем.

2.3. Создание новых отношений/таблиц базы данных

Новые таблицы создаются с помощью операторов языка SQL. При создании нового отношения в нем не будет данных. Созданное новое отношение будет содержать только строку заголовков столбцов, но не будет еще содержать никаких строк с данными.

Для выполнения операторов SQL надо в окне информации о базе данных, показанном на рис. 2.18 нажать кнопку «**View**». При этом откроется окно со статистической информацией о базе данных показанное на рис. 2.21. Эту информацию всегда можно получить по кнопке «**Overview**» в нижнем меню окна.

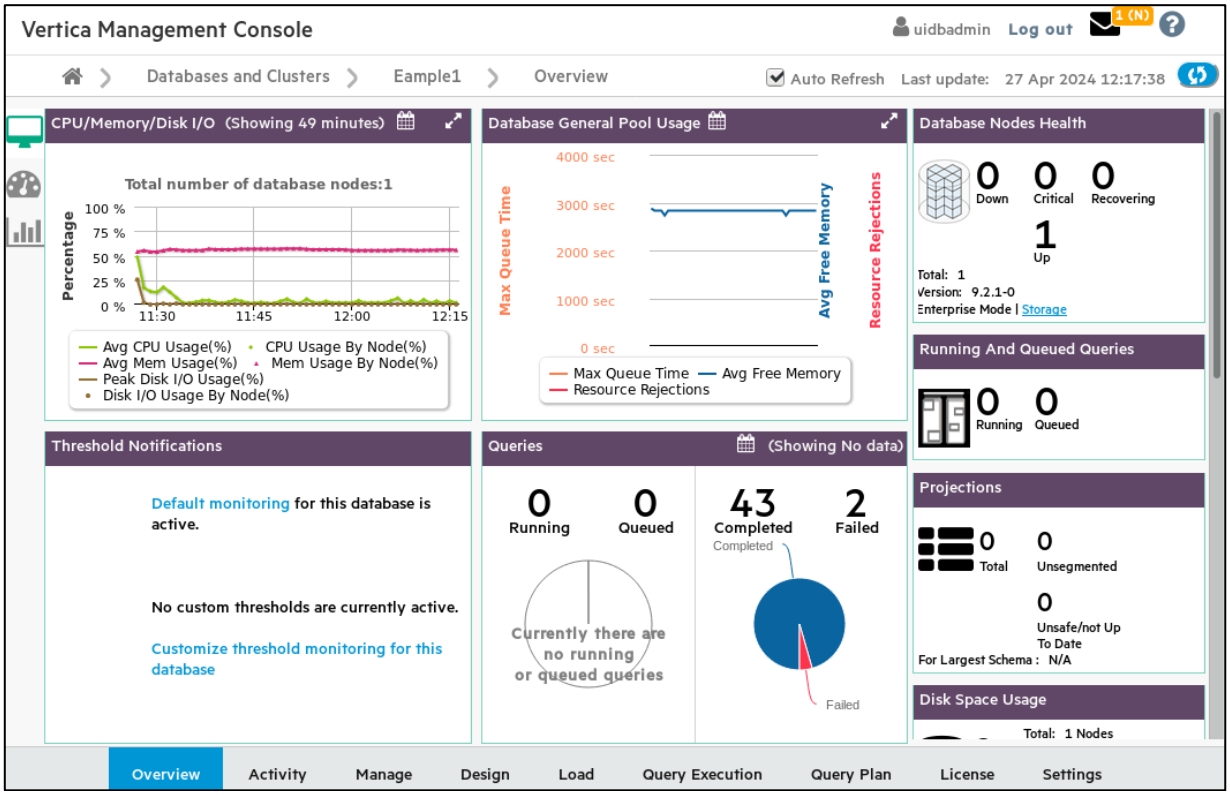


Рисунок 2.21 – Статистическая информация о базе данных

Для выполнения оператора SQL надо зайти в пункт нижнего меню “**Query Execution**” (Выполнение запросов), как показано на рис. 2.22.

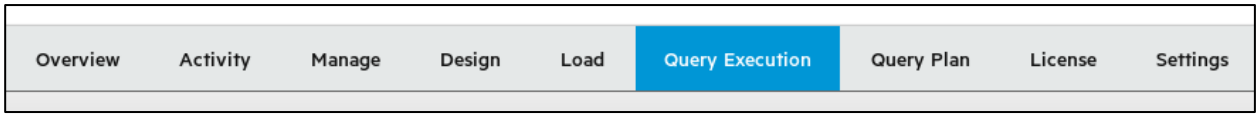


Рисунок 2.22 – Пункт меню выполнения операторов SQL

При нажатии на эту кнопку открывается окно, показанное на рис. 2.23.

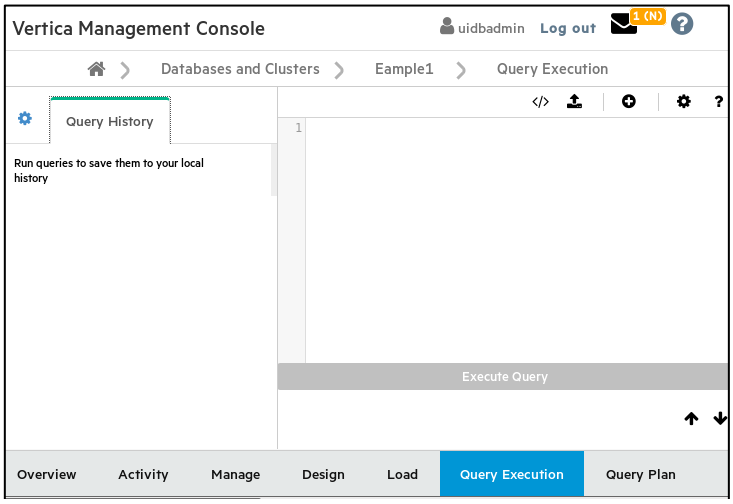


Рисунок 2.23 – Окно для набора операторов SQL

Операторы SQL набираются в виде текста в выделенном для этого поле.

В качестве примера создания отношений базы данных рассмотрим классический пример базы данных поставок товаров [11].

Пусть база данных состоит из трех отношений: «Поставщики», «Продукт» и «Поставки». Задание отношения базы данных в рассматриваемом примере включает по одному предложению языка SQL: CREATE TABLE (СОЗДАТЬ ТАБЛИЦУ) для каждой из трех составляющих ее таблиц. CREATE TABLE представляет собой пример предложения *определения данных* языка SQL. Каждое предложение CREATE TABLE специфицирует имя таблицы, которая должна быть создана, имена ее столбцов и типы данных для этих столбцов. Синтаксис оператора CREATE TABLE приведен в приложении В.

Создание таблицы/отношения «Поставщики». Идентификатор создаваемой новой таблицы будет S. В сформированном программном Таблица S представляет информацию о поставщиках. Каждый поставщик описывается следующими данными: номер, уникальный для этого поставщика, фамилию - не обязательно уникальную, значение рейтинга или состояния и местонахождение (город). Для целей примера предположим, что каждый поставщик находится в точности в одном городе. Оператор SQL для этого отношения имеет следующий вид:

CREATE TABLE S (IdS char(5), name char(20), stat int, City char(15));

Для каждой единицы данных задается ее имя и способ представления в базе данных. Например, IdS – уникальный идентификатор поставщика (Ключ), char(5) – в базе данных этот атрибут хранится в виде набора из пяти символов, name – фамилия поставщика, char(20) – символьная константа из 20 символов, stat – рейтинг поставщика, целое число (int), City – город в котором находится поставщик. На рис. 2.24 показан текст оператора в поле консоли менеджера системы Vertica. Оператор можно набирать в одну строку или в несколько строк. Чтобы была возможность скопировать оператор из файла пособия, который находится в основной машине с ОС Windows в буфер обмена виртуальной машины под ОС Linux надо настроить буфер обмена в виртуальной машине как описано в приложении А.

```
1 CREATE TABLE S (IdS char(5),
2                 name char(20),
3                 stat int,
4                 City char(15));
5
```

Рисунок 2.24 – Подготовленный оператор SQL

При нажатии на зеленое поле «**Execute query**» (Выполнить запрос) в случае правильно подготовленного текста оператора будет выдано сообщение, показанное на рис. 2.25.

Рисунок 2.25 – Сообщение об успешном выполнении оператора SQL

После выполнения оператора в левом поле окна во вкладке «**Query History**» будет записана история выполненных действий, как показано на рис. 2.26.

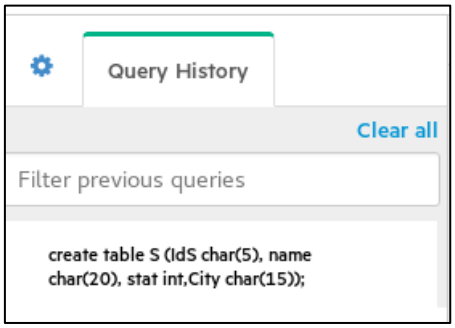


Рисунок 2.26 – История запросов к базе данных

Теперь, чтобы выполнить новый запрос можно использовать копию текста из истории запросов. Для этого надо отметить нужный фрагмент текста и воспользоваться правой кнопкой мыши, которая открывает меню действий, в котором имеется пункт **Copy** (копировать). Затем в поле запросов опять с помощью правой кнопки мыши и пункта всплывающегося меню **Paste** вставить фрагмент в формируемый запрос. На рис. 2.27 в поле запросов вставлено новое предложение с задачей создать отношение с именем S2. При этом в операторе специально с демонстрационной целью сделана синтаксическая ошибка: поставлена лишняя запятая.

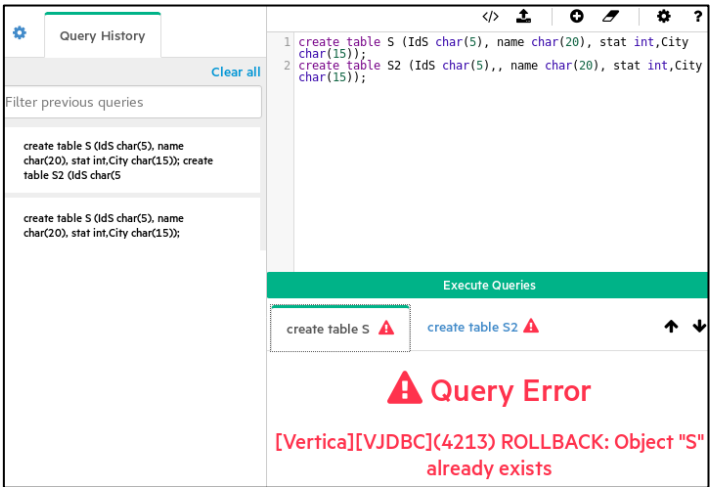


Рисунок 2.27 – Пример с ошибками в поле запросов

При нажатии на поле «**Execute Queries**» будут выполняться оба записанных оператора в поле запросов. Причем первый оператор будет выполняться повторно. В результате система нашла в обоих операторах ошибки. При выполнении первого запроса система сообщает «Object “S” already exists» (Объект «S» уже существует). Сообщение об ошибке при выполнении второго запроса показано на рис. 2.28.

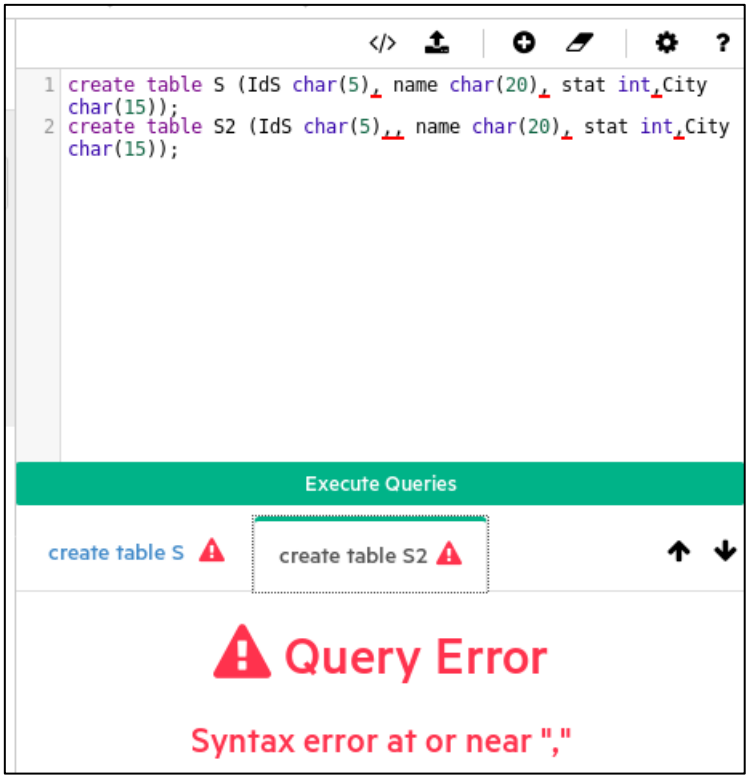


Рисунок 2.28 – Сообщение системы о синтаксической ошибке

Система сообщает, что имеет место синтаксическая ошибка, связанная с некорретным использованием запятой. При этом в тексте оператора красной чертой отмечаются возможные места ошибки.

Запросы к базе данных могут быть очень сложные. В связи с этим в системе предусмотрена возможность загрузки для редактирования и исполнения текстов запросов из файла. Для этого служит кнопка загрузки текста (скрипта) из файла с расширением SQL, показанная на рис. 2.29 (Стрелка импорта).

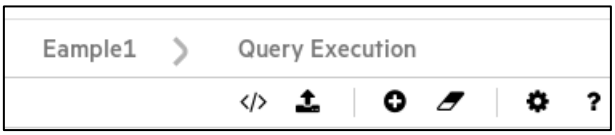


Рисунок 2.29 – Дополнительные кнопки формирования операторов

При нажатии на эту кнопку открывается содержимое папки с файлами, которые содержат сформированные заранее наборы операторов SQL. Пример

результата нажатия на кнопку показан на рис. 2.30. Кнопка меню с плюсом открывает дополнительную вкладку с окном запросов. Кнопка с резинкой очищает поля запросов кроме истории.

Для поиска

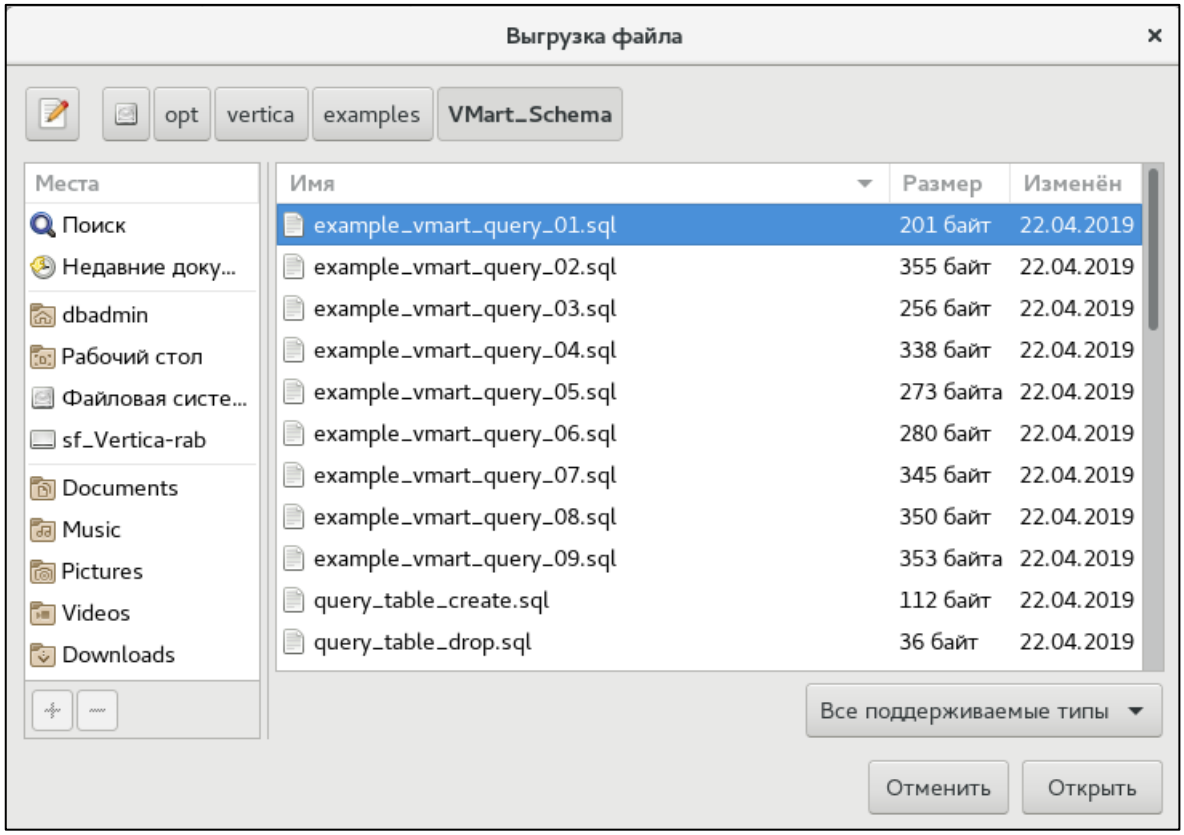


Рисунок 2.30 – Скрипты с операторами SQL

При нажатии на кнопку «**Открыть**» в проводнике текст из файла будет загружен в поле запросов.

На рис. 2.30 показаны скрипты запросов к демонстрационной базе данных. Для поиска нужного файла, который может находится в любом месте, надо выйти в главный директорий файловой системы Linux по второй слева кнопке верхнего меню. В этом случае окно с файлами будет иметь вид показанный на рис. 2.31.

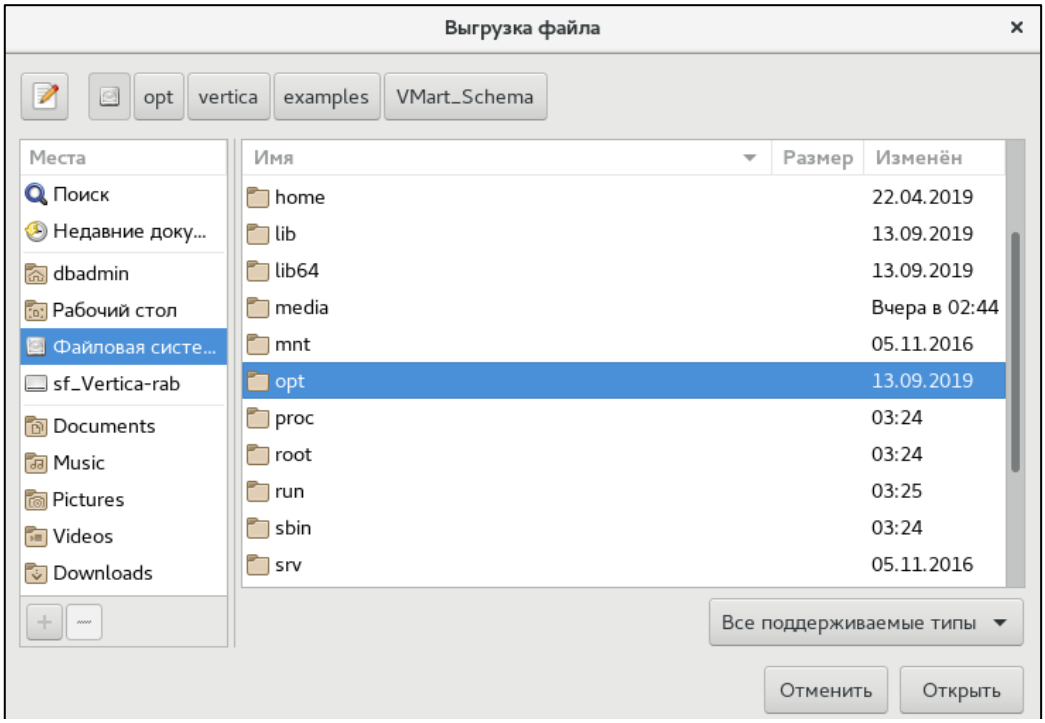


Рисунок 2.31 – Файловая система виртуальной машины

Затем надо найти файл с нужным текстом запроса к базе данных.

Упражнение 2.2. Создать таблицу поставщиков S с параметрами арибутов согласно индивидуального варианта, заданного в конце главы.

Создание таблицы «Продукт». Таблица с именем P содержит информацию о поставляемых поставщиками деталями (точнее, виды деталей). Каждый вид деталей имеет уникальный номер детали (Ключ), название детали, цвет, вес и место (город), где хранятся детали этого вида. Для целей примеров предположим, что каждый вид деталей имеет в точности один цвет и хранится на складе только одного города.

Оператор SQL для создания таблицы будет иметь следующий вид:

```
CREATE TABLE P (IdP char(5), Name char(20), Colour char(7), Weight int, City char(15));
```

На рис. 2.32 показан текст оператора в системе.

```
CREATE TABLE P (IdS char(5),
                Name char(20),
                Colour char(7),
                Weight int,
                City char(15));
```

Рисунок 2.32 – Подготовленный оператор в системе

Упражнение 2.3. Создать таблицу деталей P с параметрами арибутов согласно индивидуального варианта, заданного в конце главы.

Создание таблицы «Поставки». Таблица с именем SP представляет информацию о поставках деталей. Она служит для того, чтобы связать между собой две другие таблицы. Для каждой поставки задается номер поставщика, номер/ключ детали, и количество поставляемых деталей. Для используемого примера снова предположим, что может существовать не более одной поставки для заданного поставщика и заданной детали. Комбинация значений атрибутов НОМЕР_ПОСТАВЩИКА и НОМЕР_ДЕТАЛИ является составным ключом для определения количества поставляемых деталей.

Оператор SQL имеет следующий вид:

CREATE TABLE SP (IdS char(5), IdP char(5), Quantity int);

На рис. 2.33 показан вид оператора в системе.

```
CREATE TABLE SP (IdS char(5), IdP char(5), Quantity int);
```

Рисунок 2.33 – Оператор создания таблицы поставок

Упражнение 2.4. Создать таблицу поставок SP с параметрами атрибутов как показано на рис. 2.33.

После выполнения всех операторов в левом поле окна истории запросов будет показан список всех выполненных запросов к базе данных (операторов SQL), как показано на рис. 2.34.

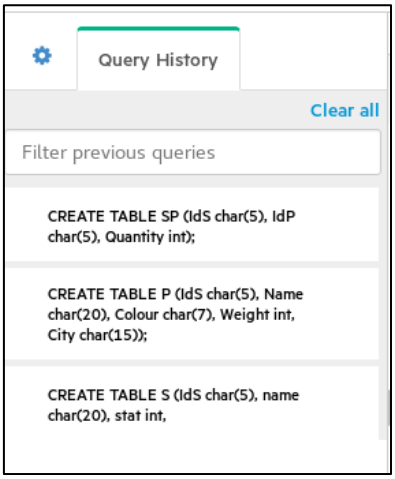


Рисунок 2.34 – История выполненных запросов

В результате, база данных состоит из трех таблиц, а именно: S, P и SP. Для просмотра состояния созданных таблиц надо войти в пункт нижнего меню “**Activity**”, как показано на рис. 2.35.

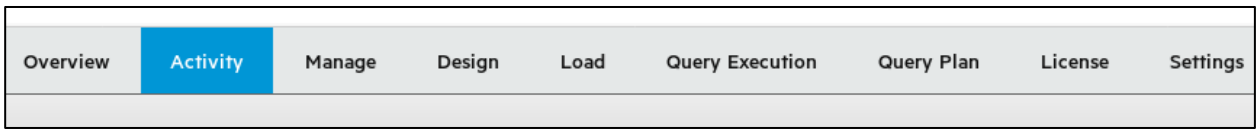


Рисунок 2.35 – Пункт меню “Activity”

В верхнем меню имеется открывающийся список возможных действий «**Queries**». В данном списке (Рис. 2.35) надо выбрать пункт “**Table Utilization**” (Использование таблиц).

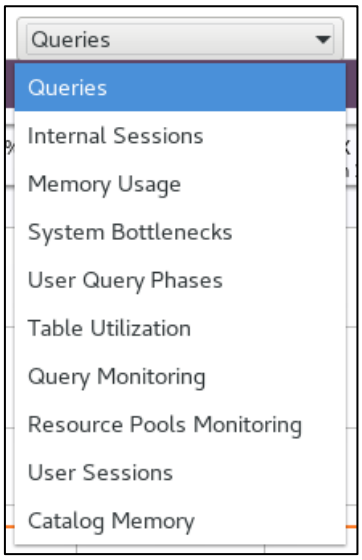


Рисунок 2.36 – Список видов информации

В результате будет открыто окно, показанное на рис. 2.37.

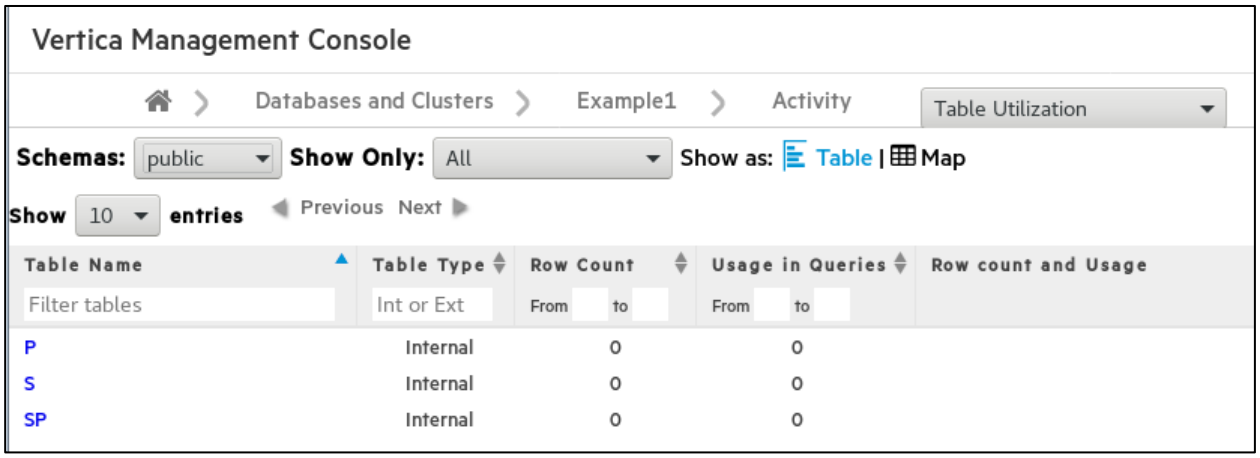


Рисунок 2.37 – Список созданных таблиц

Нажатием мышки на имени таблицы открывается поле с информацией о таблице, как показано на рис. 2.38.

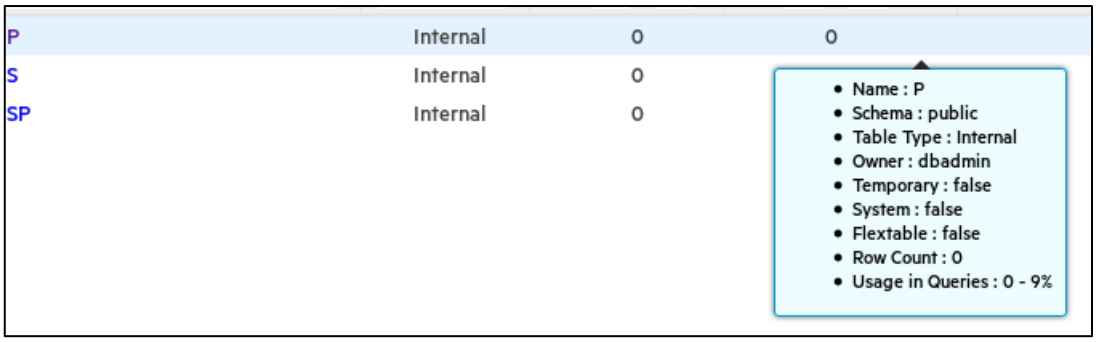


Рисунок 2.38 – Информация о таблице P

Упражнение 2.5. Просмотреть информацию о созданных базах данных.

2.4 Операторы ALTER TABLE и DROP

Имеется возможность изменить существующую таблицу, добавляя к ней справа новый столбец. Для этого используется оператор ALTER TABLE, который имеет следующий вид:

```
ALTER    TABLE    имя—базовой— таблицы
ADD      имя—столбца тип—данных;
```

Например:

```
ALTER    TABLE    S
ADD      СКИДКА SMALLINT;
```

Это предложение добавляет к таблице S столбец СКИДКА. Все существующие записи таблицы S расширяются с четырех значений полей данных до пяти, и во всех случаях новое пятое поле принимает неопределенное значение. Спецификация NOT NULL в предложении ALTER TABLE не допускается. Заметим, между прочим, что только что описанное расширение существующих записей, не означает, что в это время физически обновляются записи в базе данных. Изменяется лишь хранимое в каталоге, описание таблицы. Отдельные записи физически не изменяются до тех пор, пока они в следующий раз не станут целевыми для предложения UPDATE языка SQL (см. главу 6).

Для удаления таблицы из базы данных имеется следующий оператор языка SQL:

DROP TABLE <Имя таблицы>;

Данный оператор надо использовать в случае необходимости удаления созданной таблицы с какими то ошибками в ее структуре.

При окончании работы с базой данных желательно ее закрыть. Для этого надо войти в пункт “**Manage**”, как показано на рис. 2.39.

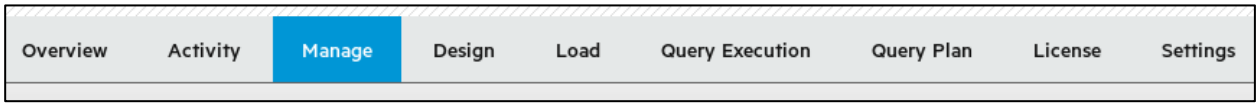


Рисунок 2.39 – Пункт меню “Manage”

Откроется окно, показанное на рис. 2.39.

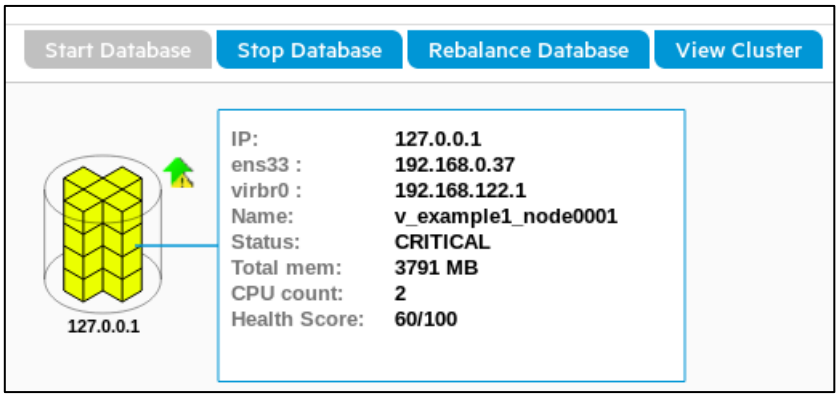


Рисунок 2.40 – Останов базы данных

Затем надо выбрать пункт меню “**Stop Database**”. Теперь обе базы данных отражаются как закрытые (Рис. 2.41).

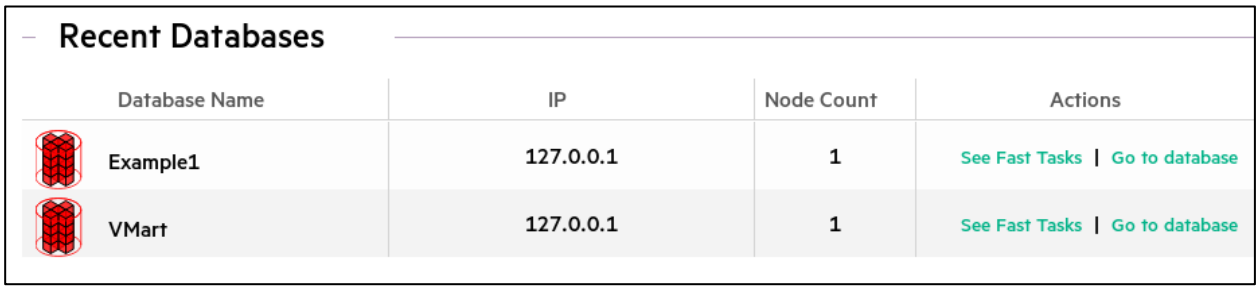


Рисунок 2.41 – Закрытые базы данных

2.5 Порядок выполнения работы

1. Загрузить виртуальную машину с системой.

2. Выполнить все упражнения.

3. Оформить отчет о проделанной работе.

2.6 Содержание отчета

Отчет должен содержать следующую информацию: вариант задания, копии окон экрана с выполненным заданием, выводы.

2.7 Контрольные вопросы

1. Что такое Big Data?
2. Что такое виртуальная машина?
3. Как войти в систему Vertica?
4. Что такое консоль менеджера?
5. Какие основные области консоли менеджера?
6. Как открыть базу данных?
7. Как закрыть базу данных?
8. Как задать имя новой базы данных?
9. Что такое отношение?
10. Что такое атрибуты отношения?
11. Что такое схема базы данных?
12. Какие Вы знаете форматы данных?

2.8 Варианты задания

Требуется создать описанную базу данных поставщиков, но с форматами данных, приведенных в следующей таблице.

Вари-ант	S.Name	S.City	P.Name	Colour	P.City
1	10	15	20	5	10
2	11	16	21	6	12
3	12	17	22	7	14
4	13	18	23	8	16
5	14	19	24	9	18
6	15	20	25	5	20
7	16	15	20	6	10
8	17	16	21	7	12
9	18	17	22	8	14
10	19	18	23	9	16
11	20	19	24	5	18
12	10	20	25	6	10
13	11	15	20	7	12
14	12	16	21	8	14
15	13	17	22	9	16
16	14	18	23	5	18
17	15	19	24	6	20
18	16	20	25	7	22
19	17	15	20	8	24
20	18	16	21	9	26

Цифры в таблице означают число символов, выделяемых для данного атрибута.

Глава 3. Лабораторный практикум. Загрузка данных в системе Vertica

Целью практикума является получение навыков в загрузке данных в СУББД. В данной работе изучаются операторы манипулирования данными INSERT (вставить), UPDATE (обновить) и DELETE (удалить) языка SQL и их реализация в системе “Vertica”. Полный синтаксис предложений приведен в приложении В.

Упражнения рассматриваются на основе созданных в предыдущей лабораторной работе трех таблиц, а именно: таблицы поставщиков S, таблицы деталей P и таблицы поставок SP.

3.1 Оператор языка SQL «INSERT»

Оператор INSERT служит для добавления в отношение новых данных и в простейшем случае имеет следующий общий формат:

```
INSERT
INTO      Имя отношения [(Имя атрибута [,Имя атрибута] . . .)]
VALUES    (константа [,константа] . . .);
```

По данной команде в созданное ранее отношение вставляется новая строка, имеющая заданные значения для указанных атрибутов, причем i-я константа в списке констант соответствует i-му атрибуту списке атрибутов. Имена атрибутов в предложении INSERT могут быть опущены. Отсутствие списка атрибутов эквивалентно спецификации списка всех атрибутов в отношении при его создании оператором CREATE. При пропуске списка атрибутов надо учитывать, что в созданном ранее отношении в СУБД порядок атрибутов может отличаться от порядка атрибутов в операторе CREATE TABLE при создании отношения. Например, атрибуты могут быть упорядочены по алфавиту. В связи с этим рекомендуется все таки задавать список атрибутов.

Упражнение 3.1. Необходимо загрузить в таблицу деталей P деталь с кодом P7, находящуюся на складе в городе 'Ялта', с весом 2 единицы. Название и цвет в настоящее время неизвестны. Предложение языка SQL будет иметь следующий вид:

```
INSERT
INTO      P (IdP, City, Weight)
VALUES    ('P7', 'Ялта', 2);
```

В базе данных создается новая запись для детали с заданным номером, городом и весом, с неопределенными значениями для названия и цвета. Эти два последних поля не должны быть, конечно, определены при создании

отношения как NOT NULL в предложении CREATE TABLE для таблицы P. Порядок слева — направо, в котором поля указаны в предложении INSERT, не обязательно должен совпадать с порядком слева — направо, в котором поля были специфицированы в предложении CREATE, так как имена атрибутов задаются явно.

Для выполнения действия на консоли менеджера базы данных надо выбрать режим “**Manage**”, как показано на рис. 3.1.

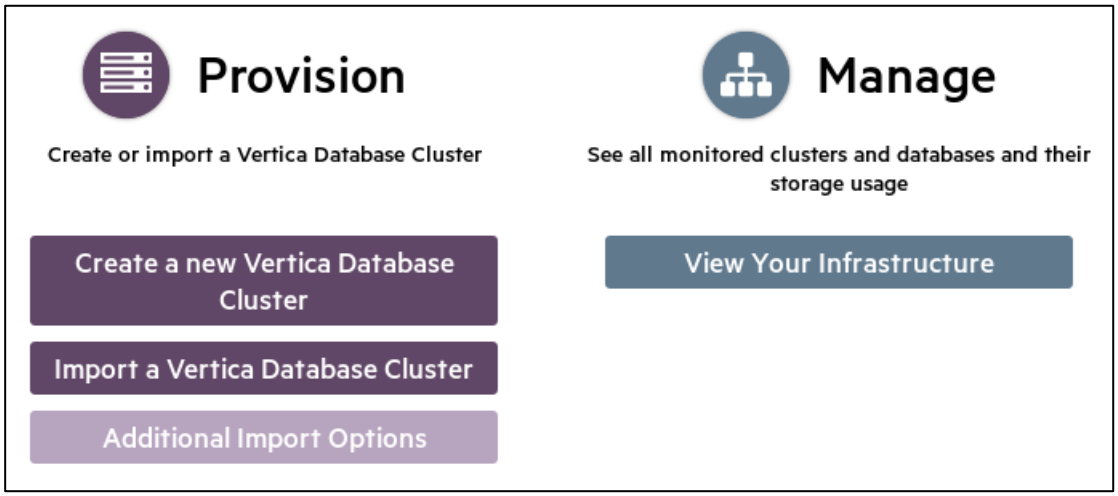


Рисунок 3.1 – Главное меню консоли менеджера

База данных должна быть открыта, как показано на рис. 3.2. Если база данных не активна, то надо остановить открытую базу данных и запустить нужную базу данных, как было описано в предыдущем практикуме. В примерах практикума рассматривается база данных Example1.

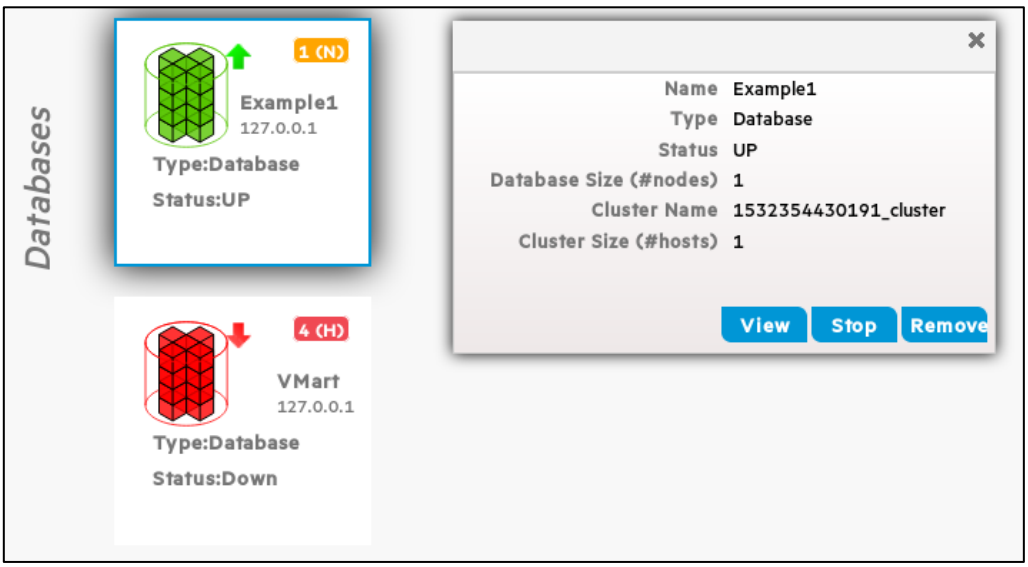


Рисунок 3.2 – Открытая новая база данных

После входа в режим просмотра базы данных “**View**”, для выполнения

операторов SQL надо зайти в пункт меню “**Query Execution**” (Выполнение запросов), как показано на рис. 3.3.

Рисунок 3.3 – Пункт меню выполнения операторов SQL

Операторы языка SQL ввода данных набираются также как это было ранее, что показано на рис. 3.4.

Рисунок 3.4 – Операторы SQL для загрузки данных

Для окончания процесса загрузки надо дополнительно выполнить оператор COMMIT. Необходимость этого действия поясняется ниже.

Для проверки выполнения действия можно выполнить простейшую форму оператора языка SQL SELECT, который в этом случае имеет вид

```
SELECT *
FROM      <Имя отношения>;
```

В результате система выдаст содержимое отношения P, как показано на рис. 3.5.

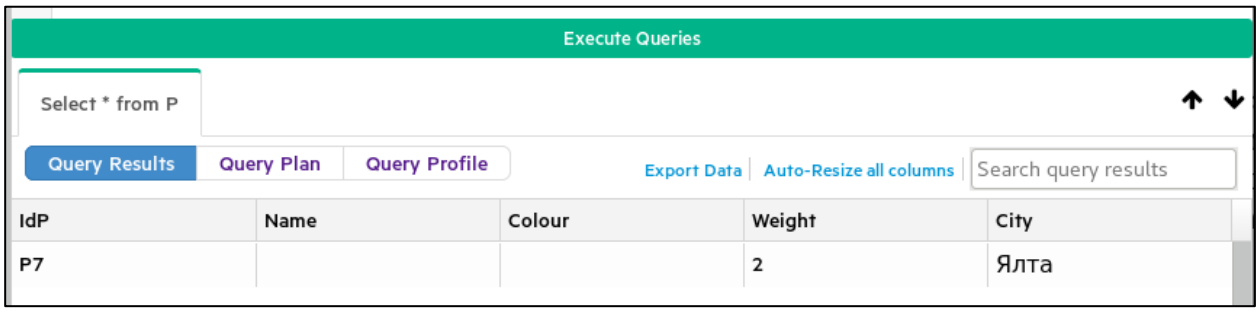


Рисунок 3.5 – Содержимое отношения P после загрузки одной строки

После того, как в отношении будут данные можно просмотреть состояние отношения в режиме Activity нижнего меню, как показано на рис. 3.6.

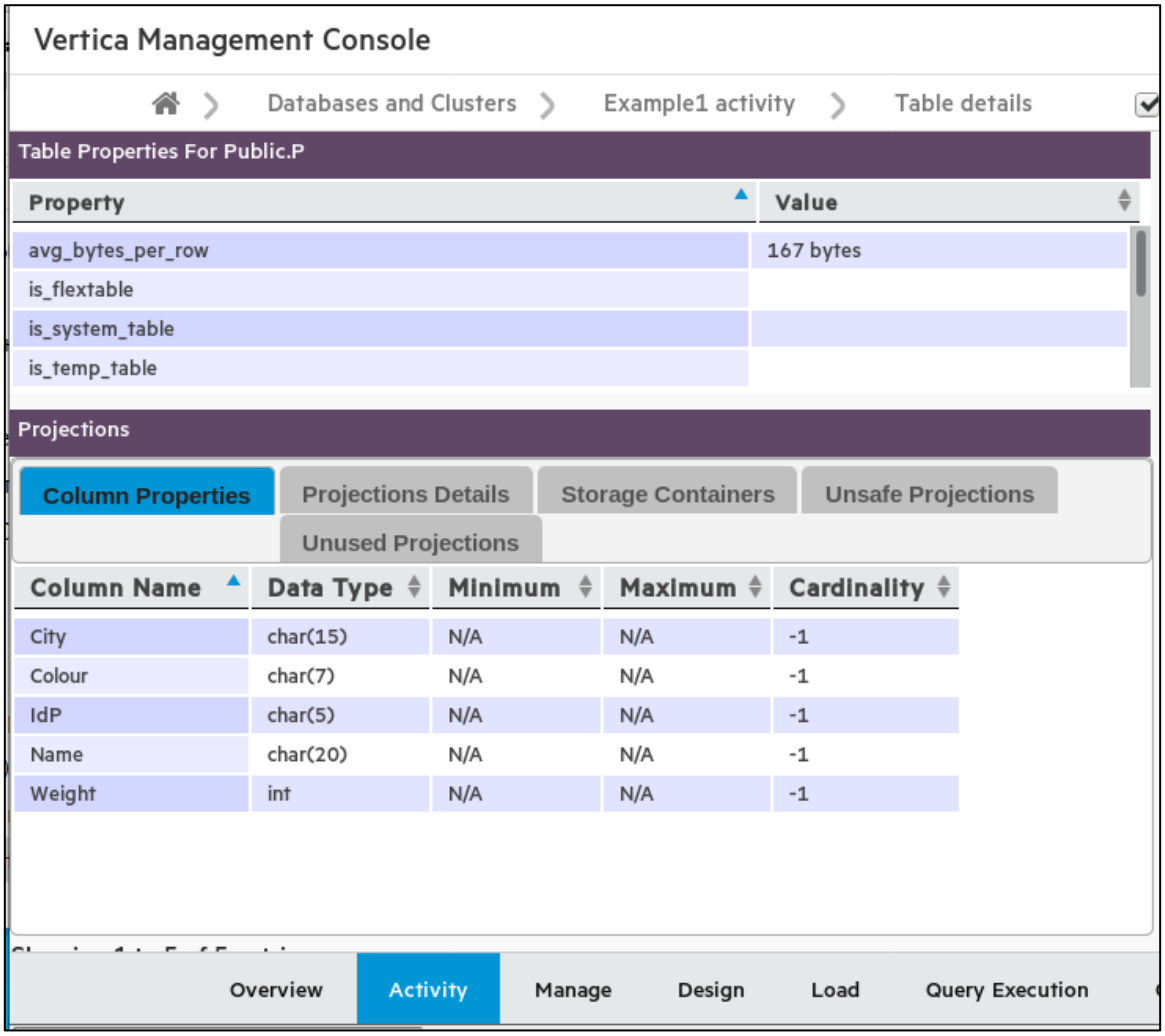


Рисунок 3.6 – Свойства отношения P

Атрибуты в отношении упорядочены по алфавиту. Именно в таком порядке их надо задавать при загрузке данных в случае умолчания списка атрибутов. Иногда при выполнении операторов с кириллицей могут возникать ошибки при работе СУБД. В связи с этим при выполнении упражнений лучше использовать латинские буквы.

Упражнение 3.2. Требуется добавить деталь с кодом P8 в таблицу P. При этом: название детали — 'Star', цвет — 'Red', вес— 4, город — 'Sevastopol'. Для недопущения потери времени из-за некоторой недоработки бесплатной версии СУББД лучше заполнять базу данных латинскими буквами.

Операторы SQL для ввода новой строки в этом случае могут иметь следующий вид:

```
INSERT
INTO      P
VALUES    ('P8', 'Star', 'Red', 4, 'Sevastopol');
```

COMMIT;

В примере отсутствие списка атрибутов эквивалентно спецификации списка всех атрибутов в отношении в порядке слева — направо, как они были определены в предложении CREATE. Такая краткая нотация может быть удобной для ввода значений всех атрибутов. Результат выполнения упражнения показан на рис. 3.7.

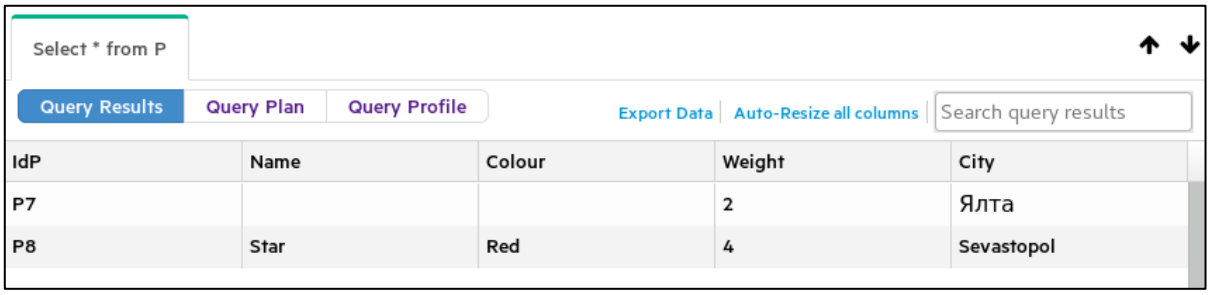


Рисунок 3.7 – Результат выполнения упражнения 3.2

Данное действие можно было выполнить и с помощью полного оператора

```
INSERT
INTO        P (IdP, Name, Colour, Weight, City)
VALUES      ('P8', 'Star', 'Red', 4, 'Sevastopol');
COMMIT;
```

Результат будет тем же самым.

Упражнение 3.3. Необходимо **в**ставить информацию о новой поставке поставщика с кодом S20, код детали P20 и количество поставляемых деталей -1000.

Операторы SQL в этом случае могут иметь следующий вид:

```
INSERT
INTO        SP (IdS, IdP, Quantity)
VALUES      ('S20', 'P20', 1000);
COMMIT;
```

Надо учитывать, что система не проверяет, имеется ли поставщик S20 в таблице S и деталь P20 в таблице P.

3.2 Оператор языка SQL “UPDATE”

Оператор UPDATE предназначен для изменения данных в отношении и имеет следующий общий формат:

```
UPDATE    <Имя отношения>
SET   Атрибут = выражение [,Атрибут = выражение] . . .
[WHERE              предикат];
```

Все записи в отношении, которые удовлетворяют условию - «предикату», обновляются в соответствии с присваиваниями «*Атрибут* = выражение» во фразе SET (установить). Предикат – это логическое выражение, созданное по правилам языка SQL. Если конструкция WHERE отсутствует, то заменяются все строки отношения.

Упражнение 3.4. Требуется в записи, созданной в упражнении 3.1, изменить цвет детали P7 на желтый. Также увеличить ее вес на 5 единиц и установить значение города «неизвестен» (NULL). Операторы SQL в этом случае имеют следующий вид:

```
UPDATE    P
SET         Colour = 'Yellow',
Weight = Weight + 5,
City = NULL
WHERE     IdP = 'P7';
COMMIT;
```

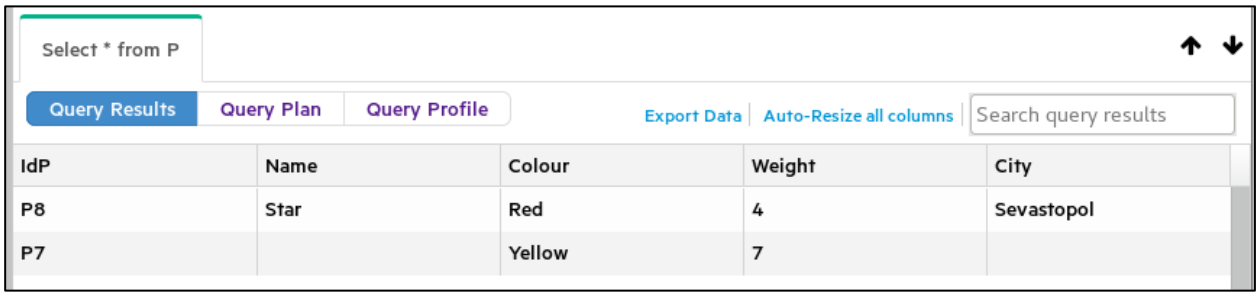


Рисунок 3.8 – Результат выполнения упражнения 3.4

Для каждой записи, которая должна быть обновлена (т. е. для каждой записи, которая удовлетворяет предикату WHERE, или для всех записей, если фраза WHERE опущена), ссылки во фразе SET на атрибуты этой записи обозначают значения этих атрибутов перед тем, как будет выполнено какое-либо присваивание в этой фразе SET. Например, предыдущее значение веса увеличивается на 5.

В системе невозможно обновить более одной таблицы в единственном запросе. Иными словами, в предложении UPDATE должна специфицироваться в точности одна таблица.

Данные в таблицы можно вставлять из файла. Эта возможность реализована в консоли операционной системы и использует команды в

режиме терминала. Подробно использование консоли рассмотрено в пятой главе и в приложении установки системы.

3.3 Оператор языка SQL «DELETE»

Оператор DELETE предназначен для удаления записей (строк таблицы) из отношения и имеет следующий общий формат:

```
DELETE
FROM    Имя отношения
[WHERE предикат];
```

Удаляются все записи в отношении, которые удовлетворяют условию - «предикату».

Упражнение 3.5. Требуется удалить запись о детали P7 из отношения P. Операторы языка SQL в этом случае имеют следующий вид:

```
DELETE
FROM       P
WHERE      IdP = 'P7';
COMMIT;
```

Теперь содержание таблицы P имеет вид, показанный на рис. 3.9.

Рисунок 3.9 – Результат выполнения упражнения 3.5

Данные удаляются только из одной таблицы. При выполнении операций удаления данных следует учитывать, что это может нарушить целостность базы данных. Так, если таблица поставок SP в настоящее время содержит какие-либо поставки удаленной детали P7 из таблицы деталей, то это удаление нарушит непротиворечивость/целостность базы данных.

3.4 Транзакции в системе Vertica

Транзакцией называется последовательность действий, которые нельзя прерывать. Система Vertica предназначена для коллективной работы

пользователей. В связи с этим в системе поддерживается механизм транзакций. При работе с консолью менеджера базы данных надо помнить, что последовательность команд, которые изменяют базу данных, представляют собой транзакцию. При реализации команд, которые изменяют базу данных, то есть добавляют строки в таблицы, изменяют данные в таблице или удаляют данные, система создает временные результаты работы. Это сделано для того, чтоб можно было осуществить отмену выполненных команд. Для окончания транзакции служит команда COMMIT. Поэтому, если не выполнить команду завершения транзакции, никакие изменения в основной базе данных не будут совершены. В этом случае после окончания сессии работы с базой данных результаты команд аннулируются.

При работе с консолью менеджера, если не указана команда COMMIT, выполнение команд изменения базы данных не приводят к результатам.

В качестве примера рассмотрим заполнение отношения «Поставщик».

Пусть требуется вставить в отношение следующие три строки, как показано на рис. 3.10.

НОМЕР_ПОСТАВЩИКА	ФАМИЛИЯ	РЕЙТИНГ	ГОРОД
S1	Иванов	20	Севастополь
S2	Петров	10	Севастополь
S3	Сидоров	30	Симферополь

Рисунок 3.10 – Три строки отношения для примера загрузки

На рис. 3.11 показаны предложения языка SQL для ввода данных в консоли менеджера системы Vertica.

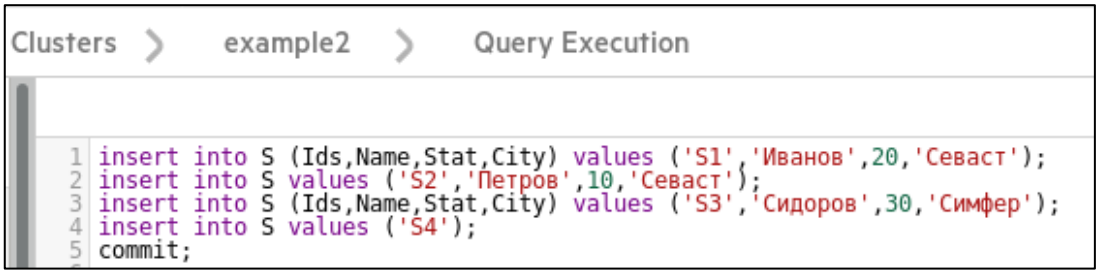


Рисунок 3.11 – Предложения SQL для ввода данных

Первый и третий операторы записаны полностью со списком атрибутов. Во втором операторе список атрибутов пропущен, так как записываются значения всех атрибутов. В четвертом операторе вводится только одно значение первого атрибута.

На рис. 3.12 показано содержание таблицы S, после загрузки данных.

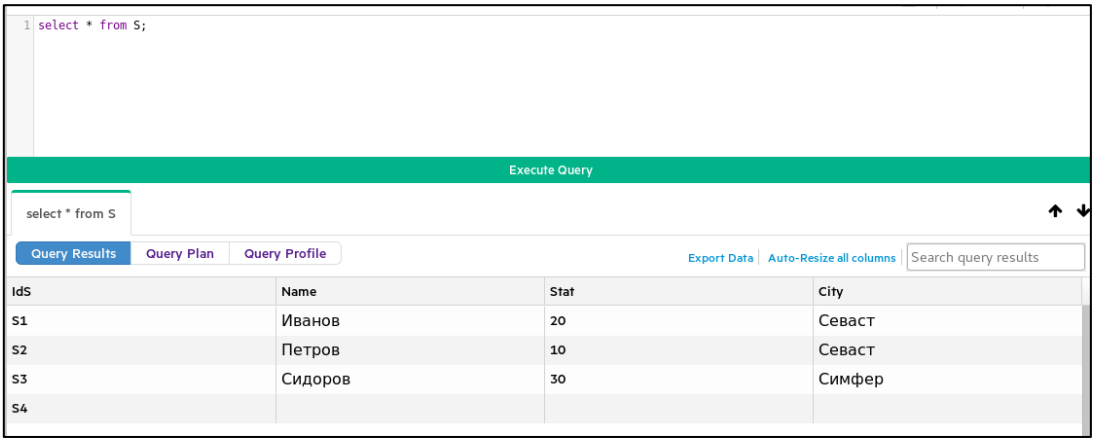


Рисунок 3.12 – Проверка содержимого отношения

На рис. 3.13 показано выполнение запроса на удаление строки с кодом поставщика S4. В список команд добавлена команда COMMIT.

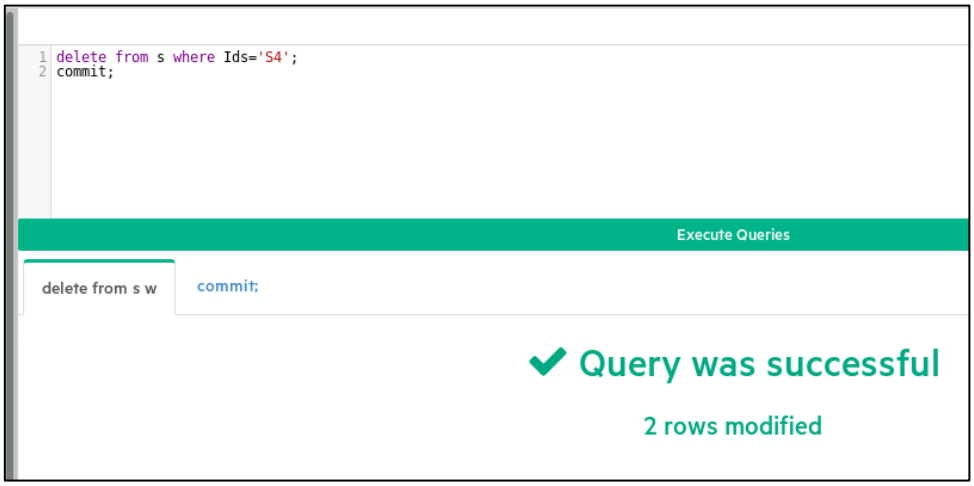


Рисунок 3.8 – Удаление строки

3.5 Порядок выполнения работы

1. Загрузить виртуальную машину с системой.
2. Выполнить упражнения.
3. Выполнить вариант индивидуального задания.
4. Оформить отчет о проделанной работе.

3.6 Содержание отчета

Отчет должен содержать следующую информацию: вариант задания, копии окон экрана с выполненным заданием, выводы.

3.7 Контрольные вопросы

1. Как загрузить запись с кодом детали и всеми неопределенными значениями?
2. Как изменить содержимое данных в базе?
3. Как удалить строки данных из таблицы?
4. Что такое «Транзакция»?
5. Зачем сразу не меняется содержимое базы данных при выполнении операторов модификации данных?
6. Какая роль у оператора COMMIT?
7. Какие существуют логические операции в предикатах?
8. Как задаются операции сравнения?
9. Что такое консоль менеджера?
10. Как открыть базу данных?
11. Как закрыть базу данных?
12. Что такое отношение?
13. Что такое атрибуты отношения?

3.8 Варианты задания

Требуется загрузить данные для базы данных поставщиков, которая состоит из следующих отношений:

Отношение «Поставщики», таблица S. Каждый поставщик имеет номер, уникальный для этого поставщика, фамилию— не обязательно уникальную, значение рейтинга или состояния и местонахождение (город). Данные для загрузки этого отношения имеют следующий вид:

НОМЕР_ПОСТАВЩИКА	ФАМИЛИЯ	РЕЙТИНГ	ГОРОД
S1	Иванов	20	Севастополь
S2	Петров	10	Севастополь
S3	Сидоров	30	Симферополь
S4	Васильева	20	Ялта
S5	Гришин	30	Москва

Оператор SQL для создания этого отношения в системе Vertica будет иметь следующий вид:

CREATE TABLE S (IdS char(5), name char(20), stat int, City char(15)).

Русские имена атрибутов и русские константы следует заменить на латинские эквиваленты. Можно использовать и свои уникальные значения.

Отношение «Продукт», таблица P. В ней представлены изготовляемые детали (товар) (точнее, виды деталей). Каждый вид деталей имеет уникальный номер детали, название, цвет, вес и место (город), где хранятся детали этого вида. Для целей примера предположим, что каждый вид деталей имеет в точности один цвет и хранится на складе только одного города. Данные для загрузки этого отношения имеют следующий вид:

НОМЕР_ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	ГОРОД
P1	гайка	красный	2	Севастополь
P2	болт	зеленый	7	Севастополь
P3	винт	голубой	7	Симферополь
P4	винт	красный	4	Ялта
P5	кулачок	голубой	2	Москва
P6	шайба	красный	9	Симферополь

Оператор SQL для создания этого отношения в системе Vertica будет иметь следующий вид:

CREATE TABLE P (IdS char(5), Name char(20), Colour char(7), Weight int, City char(15));

Здесь также русские имена атрибутов и констант следует заменить на латинские. Можно использовать и свои имена.

Отношение «Поставки», таблица SP. Она служит для того, чтобы в некотором смысле связать между собой две другие таблицы. Для каждой поставки имеется номер поставщика, номер детали, и количество деталей. Для целей примера снова предположим, что может существовать не более одной поставки в любой заданный момент времени для заданного поставщика и заданной детали. Комбинация значений НОМЕР_ПОСТАВЩИКА, НОМЕР_ДЕТАЛИ является уникальной для заданной поставки относительно множества поставок, представленных в текущий момент времени.

№	НОМЕР_ПОСТАВЩИКА	НОМЕР_ДЕТАЛИ	КОЛИЧЕСТВО
1	S1	P1	300
2	S1	P2	200
3	S1	P3	400
4	S1	P4	200
5	S1	P5	100
6	S1	P6	100
7	S2	P1	300
8	S2	P2	400
9	S3	P2	200
10	S4	P2	200
11	S4	P4	300
12	S4	P5	400

Оператор SQL для создания этого отношения в системе Vertica будет иметь следующий вид:

```
CREATE TABLE SP (IdS char(5), IdP char(5), Quantity int);
```

Русские имена атрибутов и констант следует заменить на приведенные в операторе. Можно использовать и свои имена.

Отношения S и P во всех вариантах выполнения индивидуального задания могут быть одинаковыми. В отношении SP для загрузки выбираются строки согласно вариантам, которые заданы в следующей таблице.

Вариант	Все строки таблицы SP кроме
1	1,12
2	2,11
3	3,10
4	4,9
5	5,8
6	6,7
7	6,8
8	7,9
9	8,9
10	1,9
11	2,10
12	3,8
13	4,11
14	5,12
15	6,10
16	7,12
17	8,10
18	4,12
19	5,9
20	6,8

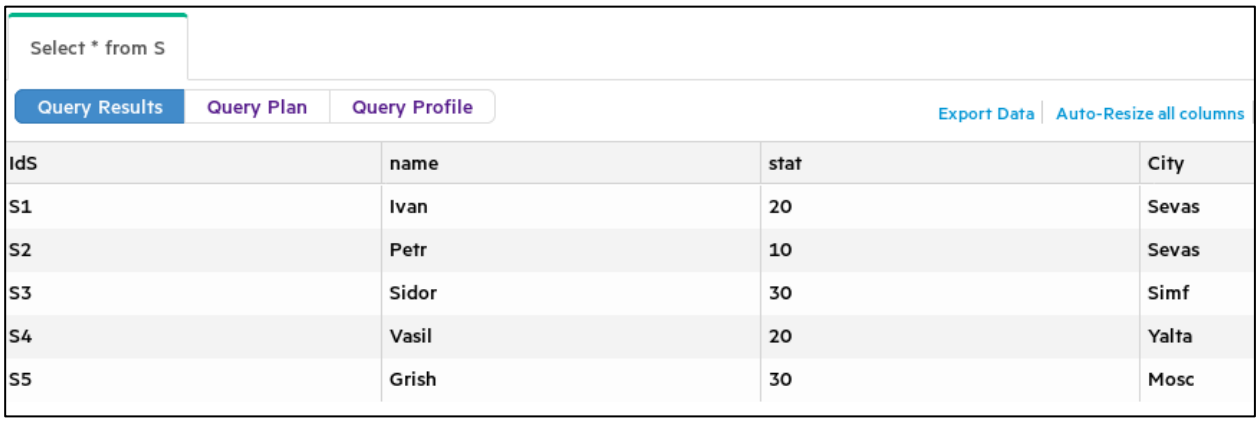
Вариант соответствует номеру студента в списке группы.

Глава 4. Запросы на получение данных в системе Vertica

В данной работе изучается формирование запросов к базе данных для поиска информации с помощью оператора языка SQL SELECT (выбрать). Выполняются упражнения на основе созданных и загруженных в предыдущих лабораторных работах трех отношений базы данных, а именно: S, P и SP.

4.1 Простые формы оператора SELECT языка SQL

Оператор SELECT выдает данные из одного или нескольких отношений базы данных. Полный синтаксис предложения приведен в приложении В. В предыдущем практическом занятии была уже использована простейшая форма этого оператора SELECT * FROM Имя таблицы. Так с помощью этого оператора можно выдать на консоль содержимое таблицы поставщиков, как показано на рис. 4.1.



IdS	name	stat	City
S1	Ivan	20	Sevas
S2	Petr	10	Sevas
S3	Sidor	30	Simf
S4	Vasil	20	Yalta
S5	Grish	30	Mosc

Рисунок 4.1 – Выдача всего содержимого таблицы поставщиков

Надо заметить, что результатом запроса является тоже таблица, которая формируется из заданных в базе данных таблиц. Полученную таблицу можно сохранить с помощью кнопки «**Export Data**» в правом верхнем углу окна результата. При нажатии на эту кнопку открывается окно браузера интранета, показанное на рис. 4.2.

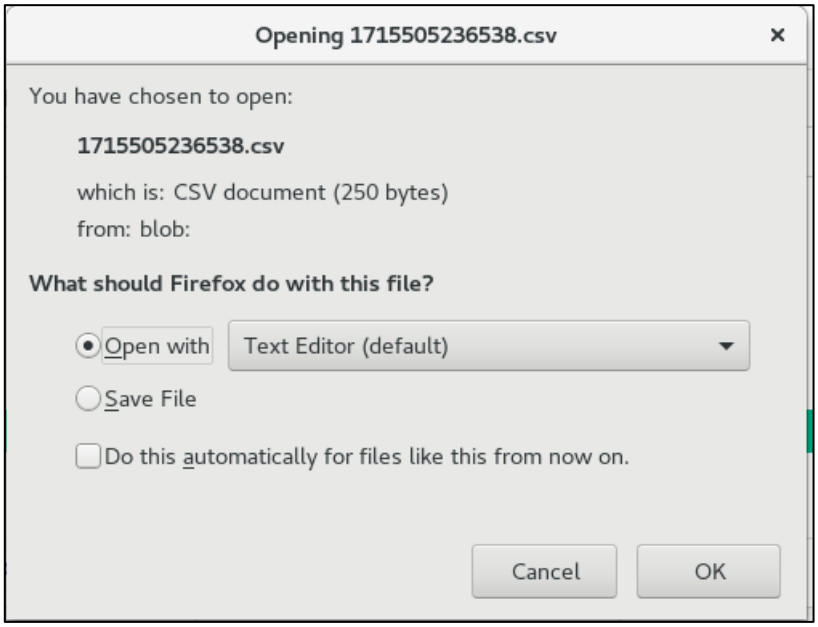


Рисунок 4.2 – Окно для сохранения данных в текстовом файле

Надо выбрать радиокнопку «**Save File**» и нажать кнопку **ОК**. В результате открывается окно, показанное на рис. 4.3.

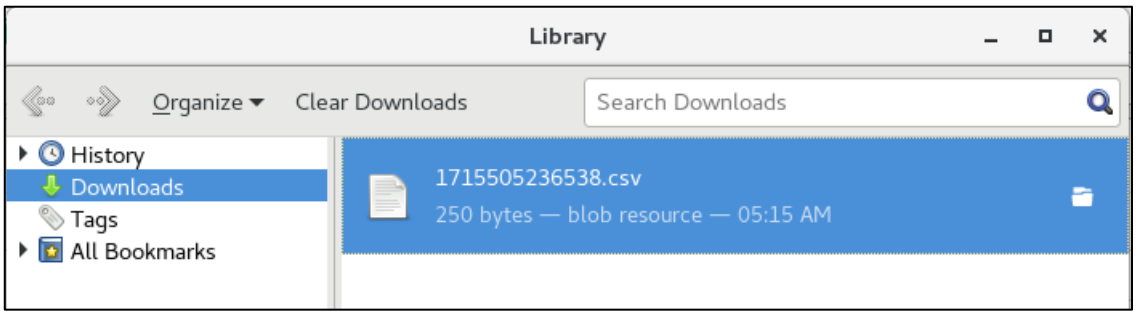


Рисунок 4.3 – Сохраненный текстовый файл с таблицей

Данные сохраняются построчно в виде текстового файла типа CSV, где значения атрибутов разделяются запятыми. Чтобы просмотреть файл в файловой системе надо найти этой файл в директории, как показано на рис. 4.4.

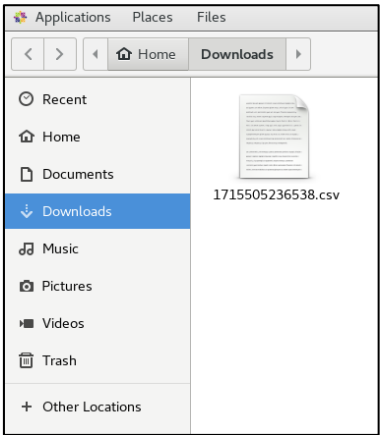


Рисунок 4.4 – Сохраненный файл в файловом менеджере ОС

Содержимое файла можно просмотреть в любом текстовом редакторе, как показано на рис. 4.5.

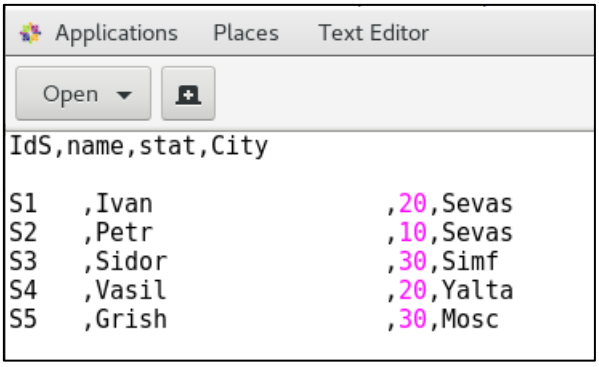


Рисунок 4.5 – Таблица в текстовом редакторе

Следует обратить внимание на то, что символьные константы занимают то число символов, которое задано при создании таблицы.

Таблицу можно преобразовать в более читабельный вид с помощью кнопки «**Auto-Resize all columns**», как показано на рис. 4.6. При этом размеры столбцов уменьшаются до более удобного вида для просмотра.

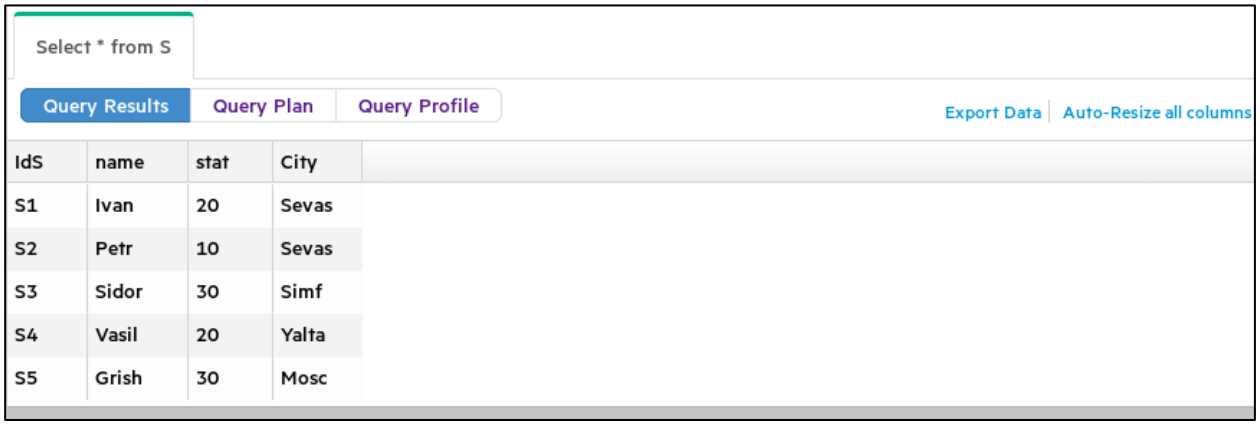


Рисунок 4.6 – Преобразованная таблица

Большинство примеров запросов в данном практикуме взяты из книги Дейта [4].

Форма оператора SELECT с условием имеет следующий вид:

```
SELECT    Список атрибутов и выражений
FROM      Список таблиц
WHERE     Условие;
```

Упражнение 4.1. Требуется получить коды и рейтинг поставщиков, находящихся в Севастополе. Таблицу получить в сжатом виде. Этот запрос может быть сформирован в СУББД Vertica следующим образом:

```
SELECT      Ids, stat
FROM        S
WHERE       City = 'Sevas';
```

Результат выполнения упражнения показан на рис. 4.7.

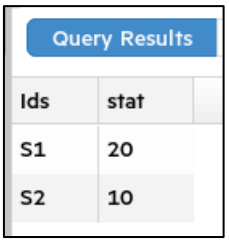
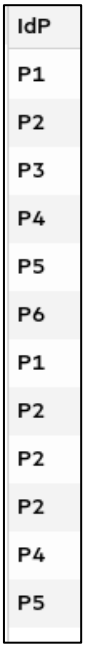


Рисунок 4.7 – Результат выполнения упражнения 4.1

Рассмотрим пример запроса на получение всех кодов деталей из таблицы поставок. Оператор SQL имеет следующий вид:

```
SELECT      IdP
FROM        SP;
```

В результате получим таблицу, состоящую из одного столбца, как показано на рис. 4.8.



IdP
P1
P2
P3
P4
P5
P6
P1
P2
P2
P2
P4
P5

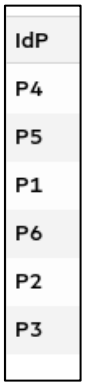
Рисунок 4.8 – Коды всех деталей из таблицы поставок

Сразу бросается в глаза, что в таблице много одинаковых строк. Для исключения одинаковых результатов запроса к базе данных в операторе SELECT предусмотрено ключевое слово DISTINCT (различный, различные).

Упражнение 4.2. Требуется получить коды всех поставляемых деталей. При этом коды не должны повторяться. Оператор SQL имеет следующий вид:

```
SELECT DISTINCT     IdP
FROM                SP;
```

В этом случае результирующая таблица показана на рис. 4.9.



IdP
P4
P5
P1
P6
P2
P3

Рисунок 4.9 – Результат выполнения упражнения 4.2

Предположим, что вес деталей задан в килограммах, а нам требуется в результирующей таблице получить вес в граммах. Для этого в операторе SELECT можно использовать арифметические выражения. Возможные арифметические и логические операции приведены в приложении В.

Упражнение 4.3. Требуется получить код детали и ее вес в граммах. Оператор SQL имеет следующий вид:

```
SELECT      IdP, Weight*1000
FROM        P;
```

В этом случае результирующая таблица показана на рис. 4.10.

IdP	?column?
P8	4000
P1	2000
P2	7000
P3	7000
P4	4000
P5	2000
P6	9000
P7	

Рисунок 4.10 – Результат выполнения упражнения 4.3

Заголовок с вычисленными значениями, которые реально не имеют столбца в исходной таблице, обозначен неопределенным именем «?column?». Также следует обратить внимание, что неопределенное значение выражения пропущено в результирующей таблице. В качестве выражения в списке вывода можно указывать константы. В используемой версии системы имеется недоработка – результирующая таблица может иметь только один неопределенный столбец. Для формирования любого имени столбца в операторе SELECT предусмотрена опция AS, которая задает имя столбца выходной таблицы.

Упражнение 4.4. Требуется модернизировать результат предыдущего упражнения таким образом, чтобы в каждой строке была символьная константа «Вес в граммах =». При этом столбец с вычисленными значениями тоже должен иметь заголовок «Вес в граммах». Оператор SQL в этом случае имеет следующий вид:

```
SELECT      IdP, 'Вес в граммах = ', Weight*1000 AS ‘Вес в граммах’
FROM        P;
```

Следует внимательно относиться к апострофам в записи символьных констант. Так оператор показанный выше даст ошибку при копировании в систему. Это связано с разным кодированием символов в разных операционных системах.

Результирующая таблица показана на рис. 4.11.

IdP	?column?	Вес в граммах
P8	Вес в граммах =	4000
P1	Вес в граммах =	2000
P2	Вес в граммах =	7000
P3	Вес в граммах =	7000
P4	Вес в граммах =	4000
P5	Вес в граммах =	2000
P6	Вес в граммах =	9000
P7	Вес в граммах =	

Рисунок 4.11 – Результат выполнения упражнения 4.4

Теперь в результирующей таблице три столбца.

4.2 Формирование условия выборки данных

За ключевым словом WHERE следует условие, или *предикат.* Предикат может включать стандартные операторы сравнения = (равно), < > (не равно), > (больше), > = (больше или равно), < (меньше), < = (меньше или равно).

Предикат формируется с помощью булевских операторов AND (и), OR (или) и NOT (нет). При этом можно использовать круглые скобки, которые указывают требуемый порядок вычисления логического выражения. Строки литер сравниваются в соответствии с их кодовым представлением. При сравнении строк литер, имеющих разные длины, более короткая строка дополняется справа пробелами.

Упражнение 4.5. Требуется получить коды поставщиков, которые находятся в Севастополе и имеют рейтинг больше, чем 20. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      IdS
FROM        S
WHERE       City ='Sevas’
AND         stat > 20;
```

Результирующая таблица показана на рис. 4.12.

SELECT IdS FROM

Query Results | Query Plan

IdS
S1

Рисунок 4.12 – Результат выполнения упражнения 4.5

В результате был получен только один код поставщика.

Для того, чтобы иметь возможность упорядочить выходные данные в таблице в операторе SELECT введена опция ORDER BY.

Для сокращения длины условия (замены два условия одним) в языке SQL введена опция BETWEEN (между).

Упражнение 4.6. Требуется получить информацию о деталях, вес которых находится в диапазоне от 4 до 10 включительно. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      *
FROM        P
WHERE       Weight BETWEEN 4 AND 10;
```

Результирующая таблица показана на рис. 4.13.

IdP	Name	Colour	Weight	City
P8	Star	Red	4	Sevastopol
P2	Bolt	Green	7	Sevas
P3	Screw	Blue	7	Simf
P4	Screw	Red	4	Yalta
P6	Washer	Red	9	Simf

Рисунок 4.13 – Результат выполнения упражнения 4.6

Может быть также задана опция NOT BETWEEN (не принадлежит диапазону между). Например, можно задать следующий оператор:

```
SELECT      *
FROM        P
WHERE       Weight NOT BETWEEN 4 AND 7;
```

Результирующая таблица показана на рис. 4.14.

IdP	Name	Colour	Weight	City
P1	Nut	Red	2	Sevas
P5	Cam	Blue	2	Mosc
P6	Washer	Red	9	Simf

Рисунок 4.14 – Результат выполнения опции Not Between

В операторе SELECT также может использоваться опция IN (принадлежит). Например, можно задать следующий оператор, чтобы получить список деталей, вес которых может быть 2, 6 или 7:

```
SELECT      *
FROM        P
WHERE       Weight IN (2, 6, 7);
```

Здесь после опции IN в скобках задается список значений, для которых ищутся строки в исходной таблице.

Результирующая таблица показана на рис. 4.15.

IdP	Name	Colour	Weight	City
P1	Nut	Red	2	Sevas
P2	Bolt	Green	7	Sevas
P3	Screw	Blue	7	Simf
P5	Cam	Blue	2	Mosc

Рисунок 4.15 – Результат выполнения опции IN

Условие IN является просто краткой записью предиката, представляющего собой последовательность отдельных сравнений, соединенных операторами OR (или). Предыдущий оператор SELECT эквивалентен следующему:

```
SELECT      *
FROM        P
WHERE       Weight = 12
OR          Weight = 16
OR          Weight = 17;
```

Упражнение 4.7. Выполните вышеприведенный запрос в системе Vertica.

В операторе SELECT также можно использовать условие NOT IN (не

принадлежит). Подобно условию IN условие NOT IN может рассматриваться только как сокращенная запись другого условия с оператором OR.

Как и во всех поисковых системах в языке ведена опция для работы с символьными строками LIKE (похоже на). Например, требуется получить список деталей, название города, в котором находится склад с деталями начинается с буквы «S». Оператор SQL в этом случае имеет следующий вид:

```
SELECT      *
FROM        P
WHERE       City LIKE 'S%';
```

Здесь символ «%» (процент) обозначает любую последовательность из символов).

Результирующая таблица показана на рис. 4.16.

IdP	Name	Colour	Weight	City
P8	Star	Red	4	Sevastopol
P1	Nut	Red	2	Sevas
P2	Bolt	Green	7	Sevas
P3	Screw	Blue	7	Simf
P6	Washer	Red	9	Simf
P7	Star	Red		Sevas

Рисунок 4.16 – Результат выполнения опции LIKE

Имеется противоположная опция NOT LIKE (не содержит).

Ниже приведено еще несколько примеров, в которых используется LIKE:

Упражнение 4.8. Требуется получить список поставщиков, в названии городов, в которых они находятся, нет буквы «е». При этом использовать конструкцию NOT LIKE ' % е %'.

Для проверки наличия (или отсутствия) неопределенного значения предусмотрен специальный предикат вида «имя — столбца» IS [NOT] NULL.

Например, надо получить список деталей, для которых не определен вес. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      *
FROM        P
WHERE       Weight IS NULL;
```

Результирующая таблица показана на рис. 4.17.

IdP	Name	Colour	Weight	City
P7	Star	Red		Sevas

Рисунок 4.17 – Результат выполнения опции IS NULL

В результате выполнения запроса был получен список из одной детали «Звездочка».

4.3 Запросы с соединением таблиц

Если требуется получить данные из нескольких таблиц, то их необходимо временно для выполнения поиска «соединить» в одну таблицу. Наличие операции соединения (join) является самым мощным средством реляционных моделей данных.

Упражнение 4.9. Требуется получить список поставщиков, которые находятся в одном городе, откуда имеются поставки деталей. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      IdS, S.name, IdP, P.Name
FROM        S, P
WHERE       S.City = P.City;
```

Следует обратить внимание на уточнение имен атрибутов в списке атрибутов и на уточнение имен атрибутов в опции WHERE. Соединение таблиц должно выполняться по одинаковым значениям атрибута City, который имеет одинаковые имена в двух разных таблицах, поэтому эти имена уточнены
во избежание двусмысленности.

Результирующая таблица показана на рис. 4.18.

IdS	name	Name
S1	Ivan	Star
S1	Ivan	Nut
S1	Ivan	Bolt
S2	Petr	Star
S2	Petr	Nut
S2	Petr	Bolt
S3	Sidor	Screw
S3	Sidor	Washer
S4	Vasil	Screw
S5	Grish	Cam

Рисунок 4.18 – Результат простого естественного соединения двух таблиц

Результирующая таблица называется *соединением* таблиц S и P по соответствию значений City. Условие S.City = P.City называется *условием соединения* или *предикатом соединения.* Вообще говоря, оператор сравнения в предикате соединения не обязательно должен быть равенством. В случае оператора равенства соединение часто называют *эквисоединением.*

Опция WHERE в SELECT-соединении может включать, помимо самого предиката соединения, также другие условия. Например, пусть в предыдущем упражнении требуется выдать данные, в которых рейтинг поставщика равен 20. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      IdS, S.name, IdP, P.Name
FROM        S, P
WHERE       S.City = P.City
AND stat = 20;
```

Следует обратить внимание на тот факт, что атрибут stat в условии задан без уточнения. Это связано с тем, что данный атрибут содержится только в одной таблице. Результирующая таблица показана на рис. 4.19.

IdS	name	Name
S1	Ivan	Star
S1	Ivan	Nut
S1	Ivan	Bolt
S4	Vasil	Screw

Рисунок 4.19 – Результат соединения двух таблиц с условием

Соединять можно любое количество таблиц.

Упражнение 4.10. Требуется получить список поставщиков и деталей, которые расположены в одном городе. Например, поставщик Sl поставляет деталь P1 и эта деталь находится на складе в том же городе. Поставщик Sl находится в Севастополе, а деталь P1 хранится также в Севастополе. Для этого требуется соединить три таблицы. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      S.IdS, P.IdP, S.City
FROM        S, SP, P
WHERE       S.IdS = SP.IdS
AND     SP.IdP = P.IdP
AND S.City=P.City;
```

Результирующая таблица показана на рис. 4.20.

IdS	IdP	City
S1	P1	Sevas
S2	P1	Sevas
S1	P2	Sevas
S2	P2	Sevas
S4	P4	Yalta

Рисунок 4.20 – Результат соединения трех таблиц с условием

Таблицу можно соединять саму с собой.

Упражнение 4.11. Требуется получить список поставщиков, которые находятся в одном городе. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      One.IdS,Two.Ids, One.City
FROM        S One, S Two
WHERE       One.City = Two.City;
```

Ясно, что в этом запросе требуется соединение таблицы S с ней самой по соответствию городов. Поэтому таблица S дважды указывается в опции FROM. Для того чтобы различать эти два ее вхождения, вводится в этом операторе два произвольных ее *псевдонима,* One и Two, и используются как уточнители в опциях SELECT и WHERE. Результирующая таблица показана на рис. 4.21.

IdS	Ids	City
S1	S2	Sevas
S1	S1	Sevas
S2	S2	Sevas
S2	S1	Sevas
S3	S3	Simf
S4	S4	Yalta
S5	S5	Mosc

Рисунок 4.21 – Результат соединения таблицы с самой собой

Полученный результат имеет строки, где показано, что поставщик находится в одном городе сам с собой. Чтобы исключить этот эффект соединения можно добавить дополнительное условие. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      One.IdS,Two.Ids, One.City
FROM        S One, S Two
WHERE       One.City = Two.City
AND One.IdS < Two.IdS;
```

Результирующая таблица показана на рис. 4.22.

IdS	Ids	City
S1	S2	Sevas

Рисунок 4.22 – Результат соединения таблицы с самой собой

При этом была также удалена строка, в которой поставщики были просто переставлены местами за счет упорядочивания по условию «меньше».

В этом примере показана техника введения синонимов для объектов базы данных.

4.4 Подзапросы

Подзапрос представляет собой оператор SELECT, который вложен в опцию WHERE другого оператора SELECT. Как правило, подзапросы используются для представления множества значений в каком-либо предикате IN.

Пусть требуется получить фамилии поставщиков, которые поставляют деталь P2. Оператор SQL в этом случае может иметь следующий вид:

```
SELECT      name
FROM        S
WHERE       IdS IN
            (SELECT     IdS
            FROM        SP
            WHERE       IdP = 'P2');
```

Результирующая таблица показана на рис. 4.23.

name
Ivan
Petr
Sidor
Vasil

Рисунок 4.23 – Результат простого подзапроса

Оператор работает следующим образом. При обработке полного запроса система обрабатывает прежде всего вложенный подзапрос. Этот подзапрос возвращает множество номеров поставщиков, которые поставляют деталь P2, а именно множество ('S1', 'S2', 'S3', 'S4').

Первоначальная задача — «Выдать фамилии поставщиков, которые поставляют деталь P2» — может быть эквивалентным образом выражена как запрос с использованием *соединения:*

```
SELECT      S. name
FROM        S, SP
WHERE       S. IdS = SP.IdS
AND         SP. IdP = 'P2';
```

Обе формулировки первоначального запроса, одна из которых использует подзапрос, а другая — соединение, в равной степени корректны. Различие только в скорости получения результата.

Упражнение 4.12. В этом упражнении рассматривается пример подзапроса с несколькими уровнями вложенности. Требуется получить фамилии поставщиков, которые поставляют по крайней мере одну красную деталь. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      name
FROM        S
WHERE       IdS IN
            (SELECT IdS
            FROM    SP
            WHERE   IdP IN
                    (SELECT IdP
                    FROM P
                    WHERE colure = 'Red'));
```

Результирующая таблица показана на рис. 4.24.

name
Ivan
Petr
Vasil

Рисунок 4.24 – Результат двойного подзапроса

Для сортировки строк выходной таблицы по возрастанию кодов или по убыванию в языке SQL имеется опция ORDER BY (упорядочить по).

Упражнение 4.13. Требуется получить список имен поставщиков, упорядоченный по рейтингу в порядке убывания их рейтинга. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      Name, Stat
FROM        S
ORDER       BY Stat DESC;
```

Результирующая таблица показана на рис. 4.25.

Query Results

Name	Stat
Grish	30
Sidor	30
Ivan	20
Vasil	20
Petr	10

Рисунок 4.25 – Результат выполнения упражнения 4.13

При упорядочивании возможно использовать два ключевых слова ASC (возрастание) или DECS (убывание), при этом по умолчанию принимается ASC. Через запятую можно задать несколько столбцов для сортировки данных. Каждое имя столбца должно обязательно идентифицировать некоторый столбец результирующей таблицы. Поэтому, например, следующее предложение недопустимо:

```
SELECT      IdS
FROM        S
ORDER       BY City;
```

В результирующей таблице отсутствует столбец с именем City.

Можно также задавать столбцы во фразе ORDER BY номерами столбцов результирующей таблицы вместо имен столбцов», где номер столбца указывает порядковую позицию (слева направо) данного столбца в результирующей таблице запроса. Благодаря этому возможно упорядочение результата на основе «вычисляемых столбцов», которые не обладают именем.

Упражнение 4.14. Выполнить упражнение 4.3 с пересчетом веса деталей в граммах, но при этом упорядочить результат по возрастанию кода детали в рамках возрастания веса в граммах. Оператор SQL в этом случае имеет следующий вид:

```
SELECT      IdP, Weight*1000
FROM        P
ORDER       BY 2, IdP;
```

Здесь «2» ссылается на второй столбец результирующей таблицы, «1» ссылается на первый столбец результирующей таблицы. Теперь результирующая таблица имеет вид, показанный на рис. 4.26.

IdP	?column?
P7	
P1	2000
P5	2000
P4	4000
P8	4000
P2	7000
P3	7000
P6	9000

Рисунок 4.26 – Результат сортировки результирующей таблицы

Оператор можно записать используя только номера столбцов результирующей таблицы.

```
SELECT      IdP, Weight*1000
FROM        P
ORDER       BY 2, 1;
```

4.5 Порядок выполнения работы

1. Загрузить виртуальную машину с системой.
2. Выполнить упражнения.
3. Выполнить вариант индивидуального задания.
4. Оформить отчет о проделанной работе.

4.6 Содержание отчета

Отчет должен содержать следующую информацию: вариант задания, копии окон экрана с выполненным заданием, выводы.

4.7 Контрольные вопросы

1. Как упорядочить выдаваемые записи по заданным атрибутам.
2. Как выбрать данные по диапазону значений.
3. Как используется список отбора.
4. Какие выражения могут использоваться в списке вывода.
5. Как получить итоговые значения.

6. Какими способами можно получить данные, находящиеся в разных таблицах.
7. Какие существуют логические операции в предикатах?
8. Как задаются операции сравнения?
9. Что такое консоль менеджера?
10. Как открыть базу данных?
11. Как закрыть базу данных?
12. Что такое отношение?
13. Что такое атрибуты отношения?

4.8 Варианты задания

Требуется выполнить следующий запрос:

Получить количество деталей, поставляемых поставщиком из города, заданного вариантом и заданного цвета. Символьные данные задать латинскими буквами. В результирующей таблицы названия столбцов должны быть на русском языке.

Вариант	Город	Цвет
1	Севастополь	Красный
2	Симферополь	Красный
3	Ялта	Красный
4	Москва	Красный
5	Севастополь	Синий
6	Симферополь	Синий
7	Ялта	Синий
8	Москва	Синий
9	Севастополь	Желтый
10	Симферополь	Желтый
11	Ялта	Желтый
12	Москва	Желтый
13	Севастополь	Зеленый
14	Симферополь	Зеленый
15	Ялта	Зеленый
16	Москва	Зеленый
17	Севастополь	Голубой
18	Симферополь	Голубой
19	Ялта	Голубой
20	Москва	Голубой

Глава 5. Построение математических моделей

В данной лабораторной работе изучаются возможности системы Vertica по аналитическому анализу больших данных.

5.1 Классический регрессионный анализ

Регрессионный анализ предназначен для установления функциональной связи между зависимой переменной (функцией отклика) *y* и независимыми переменными (факторами) x_i, i = 1,2, …m. В зависимости от числа независимых переменных анализ может быть однофакторным (одна независимая переменная) и многофакторным (две и более независимых переменных). В больших базах данных таких переменных может быть десятки тысяч.

Исходным пунктом является то, что между переменными может существовать объективная статистическая связь. На основе данных в базе надо выяснить от каких переменных зависит наблюдаемая величина y. Эту связь можно представить функцией линейной от параметров. Функциональная связь в регрессионной модели дополняется аддитивной случайной переменной ε. Регрессионное уравнение для линейной модели имеет вид:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\alpha} + \boldsymbol{\varepsilon}.$$

Здесь **Y** – вектор значений зависимой переменной в n – опытах, **X** – матрица значений факторов в тех же опытах с единичным столбцом для постоянного члена модели, **α** – вектор неизвестных коэффициентов, **ε** – вектор значений случайной составляющей.

Для оценки коэффициентов регрессионного уравнения чаще всего используется метод наименьших квадратов (МНК). Суть метода – параметры регрессионной модели оцениваются с учетом минимума суммы квадратов ошибок. Под ошибкой в данном случае необходимо понимать разницу между эмпирическим (наблюдаемым) и оцененным (рассчитанным при помощи модели) значением зависимой переменной. Формула для оценки параметров (коэффициентов) регрессионной модели при помощи МНК в матричной форме имеет вид

$$\hat{\alpha} = \left[X' \cdot X \right]^{-1} \cdot X' \cdot Y .$$

Штрих означает операцию транспонирования матрицы, крышечка над вектором параметров означает не само значение параметра, а его оценку.

Анализ классической модели регрессии состоит из двух этапов. Первый этап – это проверка значимости коэффициентов регрессионного уравнения. Второй этап – проверка модели на адекватность.

При проверке значимости регрессионного коэффициента необходимо выполнить следующие шаги:

- Сформировать гипотезы:

$H_0 : \alpha_i = 0$ - коэффициент не значим; $H_1 : \alpha_i \neq 0$ - коэффициент значим.

- Выбрать уровень ошибки первого рода (уровень значимости) α.

- Определить табличное значение t-критерия Стьюдента с учетом уровня значимости α и числа степеней свободы $(n-k)$, где n - число опытов, k - число параметров модели (в данном случае используется двухсторонний тест).

- Рассчитать экспериментальное значение t критерия Стьюдента по формуле

$$t_{p_{\alpha_i}} = \frac{|\hat{\alpha}_i|}{\sqrt{\hat{\sigma}^2_{\alpha_i}}},$$

где $\hat{\alpha}_i$ - оцененное значение i параметра (коэффициента), $\hat{\sigma}^2_{\alpha_i}$ - оценка дисперсии параметра $\hat{\alpha}_i$. Это i элемент, расположенный на главной диагонали ковариационной (дисперсионной) матрицы оценок параметров модели (коэффициентов). Дисперсионная матрица рассчитывается по формуле:

$$S_{\hat{\alpha}_i} = \hat{\sigma}^2_u \cdot [X' \cdot X]^{-1},$$

где $S_{\hat{\alpha}_i}$ - дисперсионная матрица, $\hat{\sigma}^2_{\xi}$ - оценка дисперсии случайной составляющей ε. Оценка дисперсии случайной составляющей определяется следующим образом:

$$\hat{\sigma}^2_{\xi} = \frac{RSS}{n-k},$$

где RSS - остаточная сумма квадратов отклонений.

- Если расчетное значение критерия Стьюдента больше, чем табличное значение, то с учетом уровня значимости не отвергают альтернативную гипотезу о значимости параметра модели.

Следует отметить, что t-тесты обеспечивают эффективную проверку предельного вклада каждой переменной при допущении, что все другие переменные уже включены в уравнение.

Для проверки адекватности всей модели используется критерий Фишера для оценки того, значимо ли объяснение, даваемое уравнением в целом. При проверке модели на адекватность необходимо выполнить следующие шаги:

- Сформулировать гипотезы:

H_0 : $\alpha_i = 0$ для всех i - модель не адекватна; $H_1 : \alpha_1 \neq \alpha_2 \neq \ldots \neq \alpha_i \neq 0$ - модель адекватна.

- Выбрать уровень значимости α.

- Определить табличное значение F критерия Фишера с учетом уровня значимости и числа степеней свободы $k-1$ и $n-k$.

- Определить экспериментальное значение F критерия Фишера по формуле:

$$F_p = \frac{R^2/(k-1)}{(1-R^2)/(n-k)},$$

где R^2 - коэффициент детерминация. Коэффициент детерминации – это та часть вариации признака (зависимой переменной), которая объясняется уравнением регрессии. $1-R^2$ - ошибка модели или та часть вариации признака, которая не объясняется уравнением регрессии.

Коэффициент детерминации определяется при помощи следующей формулы:

$$R^2 = \frac{\hat{\alpha}' \cdot X' \cdot Y - n \cdot \bar{Y}^2}{Y' \cdot Y - n \cdot \bar{Y}^2},$$

где $\bar{Y}$ - среднее значение зависимой переменной.

- Если расчетное значение критерия Фишера больше, чем табличное значение, то с учетом уровня значимости не отвергают альтернативную гипотезу об адекватности модели.

Если произведена оценка параметров модели и проведен анализ уравнения регрессии, то без значительных дополнительных затрат можно рассчитать частный коэффициент детерминации ΔR_i^2 (читается: дельта-R^2). Этот коэффициент также называют предельным (граничным) вкладом i-й независимой переменной в R^2. ΔR_i^2 показывает, на какую величину уменьшится коэффициент детерминации, если i-я независимая переменная (и только она) будет исключена из группы независимых переменных. Формула для расчета частного коэффициента детерминации имеет вид:

$$\Delta R_i^2 = \frac{(1-R^2) \cdot t_i^2}{n-k},$$

где t_i^2 - квадрат вычисленного значения критерия Стьюдента для i-й независимой переменной.

5.2 Построение регрессионных моделей для больших данных

Если число переменных в модели очень большое, то информационная матрица Фишера является вырожденной и найти обратную матрицу невозможно. Это приводит к необходимости использовать специальные

методы для решения задачи поиска оценок коэффициентов регрессионной модели.

Метод Ридж-регрессии вводит в функционал критерия оптимизации специальный штрафной коэффициент. В системе Vertica этот метод регуляризации обозначен как L2.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Коэффициент λ можно регулировать. Так при λ = 0 метод преобразуется в обычный метод наименьших квадратов для классической регрессии.

Метод Lasso Regression (Least Absolute Shrinkage and Selection Operator) добавляет штраф в виде суммы абсолютных значений коэффициентов

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

В системе Vertica этот метод регуляризации обозначен как L1. Оба метода различаются селекцией оставшихся переменных в модели.

Метод ENet (Elastic Net) является промежуточным между двумя описанными выше. Здесь добавляются два штрафа.

$$\boldsymbol{\beta}_{\text{ElasticNet}} = \min_{\boldsymbol{\beta}} RSS + \lambda \left[\underbrace{(1-\alpha)\|\boldsymbol{\beta}\|_2^2/2}_{L_2\text{-penalty}} + \underbrace{\alpha\|\boldsymbol{\beta}\|_1}_{L_1\text{-penalty}} \right]$$

Появляется еще один параметр α, который задает численную устойчивость метода (Этот параметр отличается от введенного ранее уровня значимости).

5.3 Методы решения задачи оптимизации

Наиболее известный метод поиска оптимума функции – метод Ньютона. Основная идея метода заключается в следующем: задаётся начальное приближение вблизи предположительного корня, после чего строится касательная к графику исследуемой функции в точке приближения, для которой находится пересечение с осью абсцисс. Эта точка берётся в качестве следующего приближения. И так далее, пока не будет достигнута необходимая точность. Иллюстрация метода Ньютона показана на рис. 5.1.

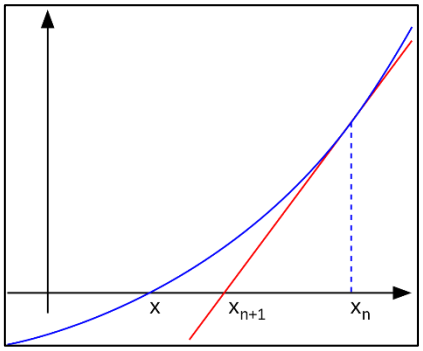


Рисунок 5.1 – Иллюстрация метода Ньютона

Синим изображена функция f(x), ноль которой необходимо найти, красным — касательная в точке очередного приближения x_n. Здесь мы можем увидеть, что последующее приближение x_{n+1} лучше предыдущего x_n.

Формула итеративного приближения может быть получена из оценки касательной:

$$f'(x_n) = \operatorname{tg}\alpha_n = \frac{\Delta y}{\Delta x} = \frac{f(x_n) - 0}{x_n - x_{n+1}} = \frac{0 - f(x_n)}{x_{n+1} - x_n},$$

Искомое приближение может быть выражено в виде формулы

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

На основании метода Ньютона были разработаны так называемые квази-ньютоновские методы — методы оптимизации, основанные на накоплении информации о кривизне целевой функции по наблюдениям за изменением градиента, чем принципиально отличаются от ньютоновских методов.

В системе Vertica кроме метода Ньютона используются следующие квази-ньютоновские методы поиска оптимального решения.

Метод Broyden–Fletcher–Goldfarb–Shanno (BFGS).

Метод покоординатного градиентного спуска Coordinate Gradient Descent (CGD).

5.4. Линейная регрессионная модель в системе Vertica

Аналитические функции в системе введены как функции над элементами таблиц (отношений). Функция построения линейной регрессионной модели имеет следующий синтаксис:

```
LINEAR_REG ( 'model-name', 'input-relation', 'response-column', 'predictor-columns'
[ USING PARAMETERS
[exclude_columns='excluded-columns']
```

```
[, optimizer='optimizer-method']
[, regularization='regularization-method']
[, epsilon=epsilon-value]
[, max_iterations=iterations]
[, lambda=lamda-value]
[, alpha=alpha-value] ] )
```

Параметры оператора показаны в следующей таблице.

model-name	Название/имя модели. Должно быть уникальным в активной схеме базы данных.
input-relation	Имя отношения или взгляда с исходными данными.
response-column	Имя атрибута с функцией отклика.
predictor-columns	Список через запятую атрибутов входных переменных/факторов. Можно использовать * для всех атрибутов. В этом случае необходимо задать атрибут с выходной переменной в параметре *excluded-columns.*
`exclude_columns`	Список через запятую атрибутов, которые надо исключить из модели.
`optimizer`	Используются следующие методы построения модели (оптимизации): Newton BFGS CGD Для метода CGD, *regularization-method* должен быть выбран L1 или ENet, иначе будет выдано сообщение об ошибке. **По умолчанию**: Задается CGD если *regularization-method* L1 или ENet, иначе Newton.
`regularization`	Методы регуляризации. None (по умолчанию) L1 L2 ENet
`epsilon`	Параметры точности построения модели. **По умолчанию**: 1e-6
`max_iterations`	Максимальное число итераций при поиске значений коэффициентов. **По умолчанию**: 100
`lambda`	Параметр регуляризации. Должен быть равным 0 или положительным. По умолчанию: 1
`alpha`	Параметр для ENet

5.5 Работа в командной консоли администратора

Многие аналитические функции в системе Vertica можно выполнить только в командной консоли операционной системы Linux. В этом режиме все действия по работе с базой данных будут производиться в консоли, т.е. в текстовом режиме.

Для того чтобы открыть консоль, необходимо нажать правую кнопку мыши на рабочей области экрана и выбрать пункт «**Console**», как показано на рис. 5.2.

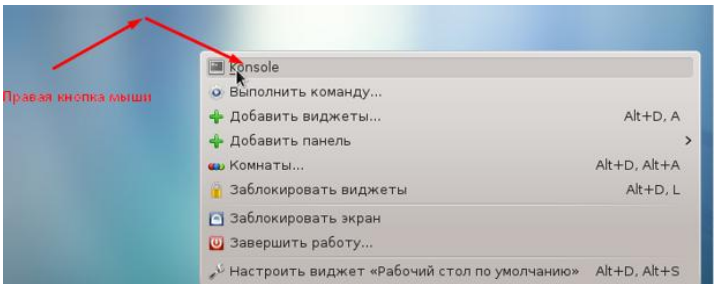


Рисунок 5.2 – Открытие консоли в режиме администратора

В другой версии операционной системы Linux может быть пункт “**Open Terminal**”, как показано на рис. 5.3.

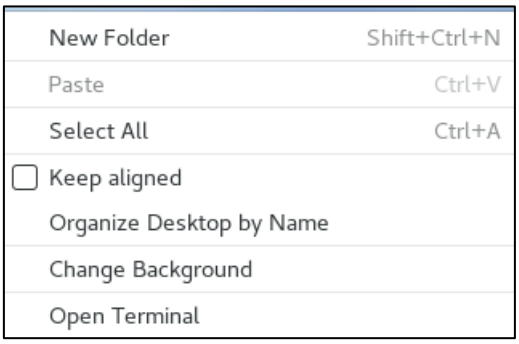
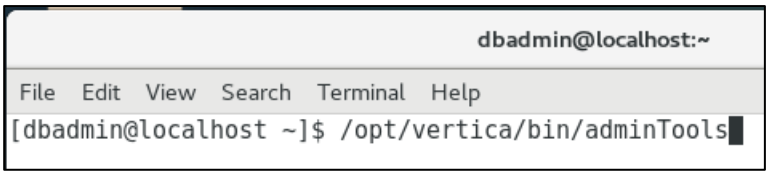


Рисунок 5.3 - Окно выбора команды

Затем нужно для вызова системы ввести в командной строке консоли после знака доллара следующий текст: /opt/vertica/bin/adminTools, как показано на рис. 5.4.



.

Рисунок 5.4 – Запуск системы Vertica в консоли

Данное действие приведет к запуску системы Vertica в консоли и появлению главного меню системы, как показано на рис. 5.5.

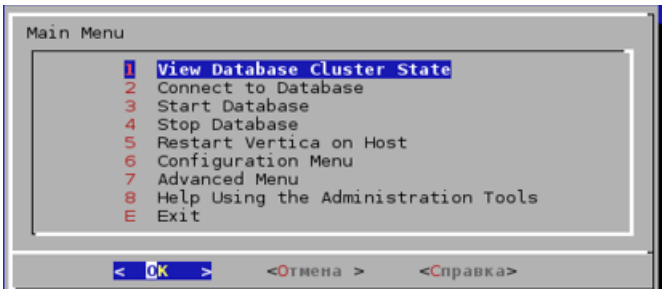


Рисунок 5.5 – Главное меню системы

Для создания новой базы данных необходимо выбрать пункт меню «**Configuration Menu**» посредством нажатия вертикальной стрелки « » на клавиатуре или с помощью мыши. После того, как данный пункт меню будет подсвечен, нажимаем кнопку «**ОК**». На экране появляется следующее меню, показанное на рис. 5.6, в котором можно выбрать пункт «**Create Database**» – «Создать базу данных».

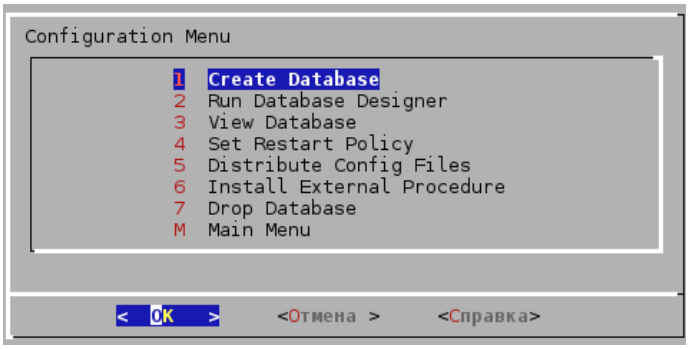


Рисунок 5.6 – Окно «Configuration Menu»

Как и при создании базы данных из браузера надо предварительно закрыть демонстрационную базу данных «VMart» или другую базу данных, которая была создана в предыдущих практикумах. Для этого надо вернуться в главное меню и выбрать пункт «**Stop Database**», как показано на рис. 5.7.

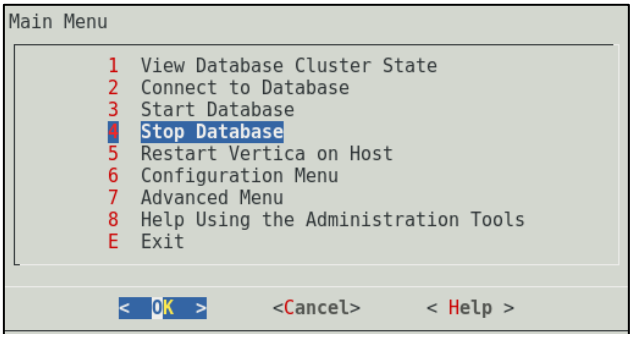


Рисунок 5.7 – Выбор пункта меню для останова базы данных

Откроется окно с выбором базы данных для остановки, как показано на рис. 5.8.

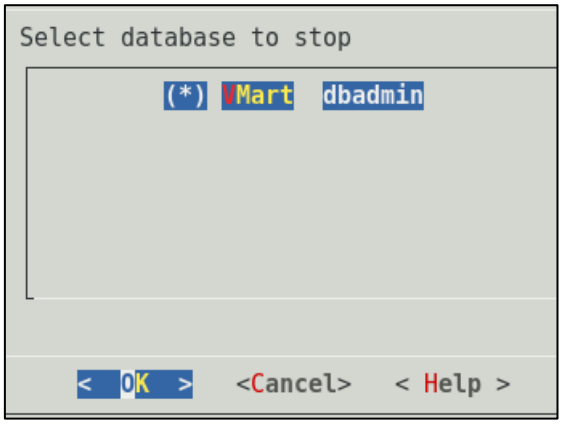


Рисунок 5.8 – Выбор базы данных для останова

Надо вставить маркер в поле с круглыми скобками и нажать кнопку «**ОК**». В результате появится окно для ввода пароля останавливаемой базы данных, как показано на рис. 5.9.

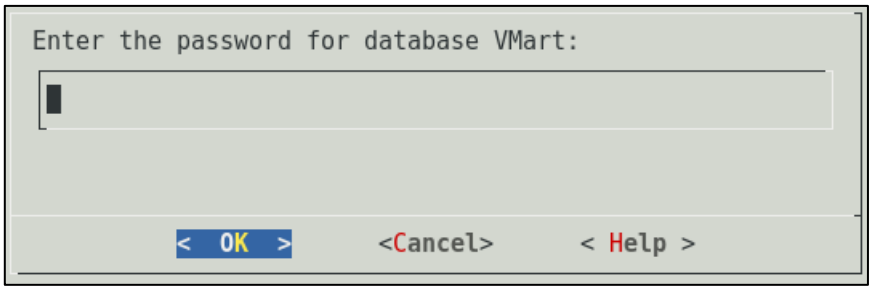


Рисунок 5.9 – Запрос пароля к останавливаемой базе данных

В используемой версии системы может появиться предупреждение, показанное на рис. 5.10.

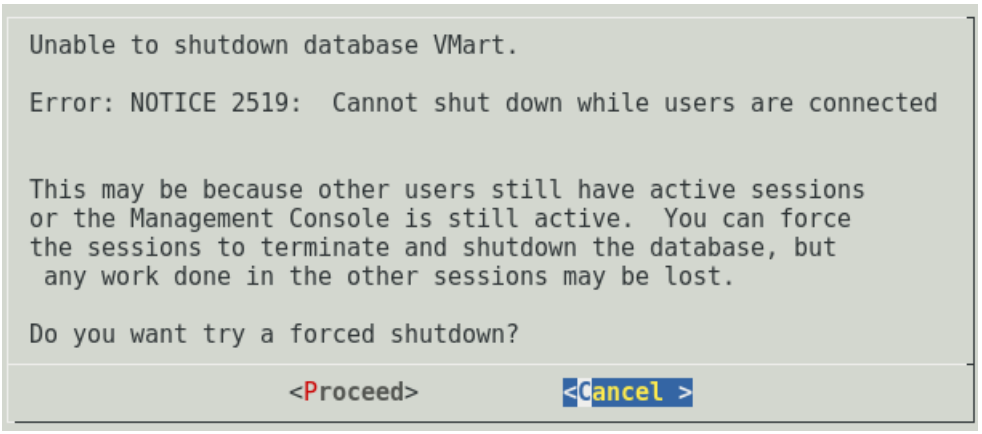


Рисунок 5.10 – Предупреждение перед остановкой базы данных

Нажимаем кнопку «**Proceed**». В результате появится сообщение об успешном останове базы данных, как показано на рис. 5.11.

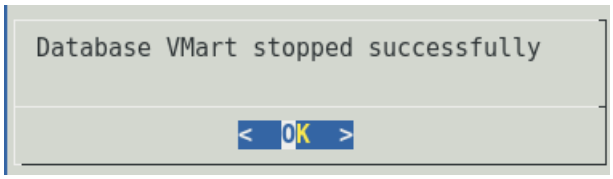


Рисунок 5.11 – Сообщение об успешном останове базы данных

После выбора пункта меню «**Create Database**» система предлагает ввести название будущей базы данных и комментарий. Пример ввода значений показан на рис. 5.12.

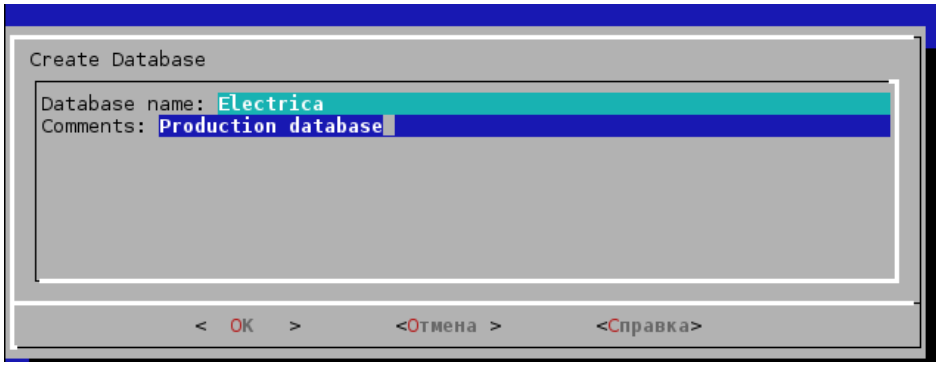


Рисунок 5.12 – Ввод названия базы данных

После нажатия кнопки «**ОК**» появляется окно, показанное на рис. 5.13. Тут необходимо ввести пароль, который будет использоваться в дальнейшем при подключении к этой базе данных. При выполнении практикума рекомендуется использовать стандартный пароль <password>.

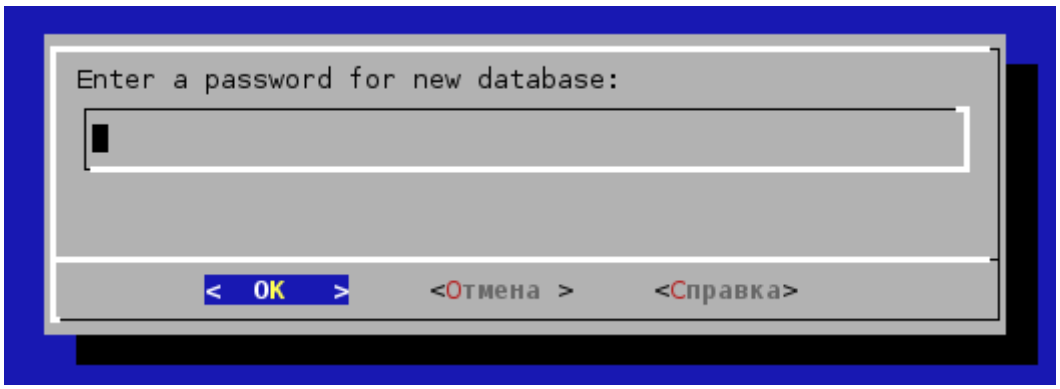


Рисунок 5.13 – Окно ввода пароля для БД

Ввести пароль и нажать «**ОК**». В качестве пароля вновь используем комбинацию «password». Во вновь появившемся окне ввести пароль еще раз для его подтверждения и снова нажать «**ОК**».

Следующее окно, которое должно появиться, показано на рис. 5.14.

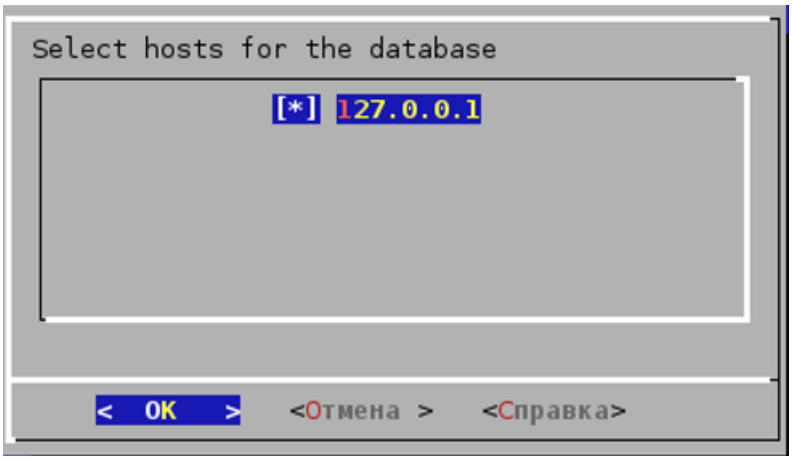


Рисунок 5.14 – Ввод адреса узла сети

Тут необходимо выбрать узел связи в сети TCP/IP. По умолчанию заносится значение 127.0.0.1 – IP-адрес, с помощью которого компьютер может обратиться по сети к самому себе, независимо от наличия у него подключения к сети. Оставить это значение и нажать «**ОК**».

Открывается окно, показанное на рис. 5.15. Тут необходимо задать папку ОС, в которой будет располагаться создаваемая база данных. Оставть без изменения.

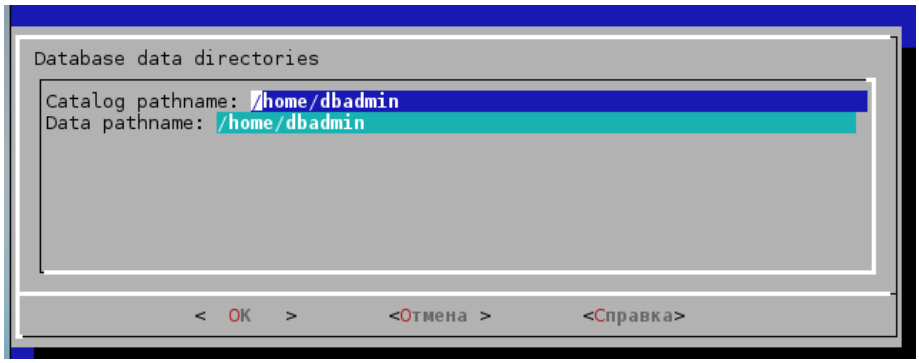


Рисунок 5.15 – Выбор каталога для хранения БД

После нажатия кнопки «**ОК**» появляется окно с предупреждением о том, что данные могут быть утрачены в случае сбоя в системе. Принять это к сведению и продолжить создание БД.

После выполнения описанных выше действий появляется окно, запрашивающее подтверждение операции создания новой базы данных и показанное на рис. 5.16.

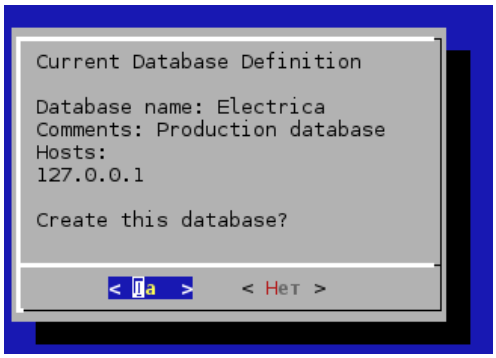


Рисунок 5.16 – Окно подтверждения создания БД

Нажимаем кнопку «**Да**» или «**Yes**» в зависимости от используемой версии ОС. Если база данных «VMart» не была закрыта, то появится сообщение, показанное на рис. 5.17.

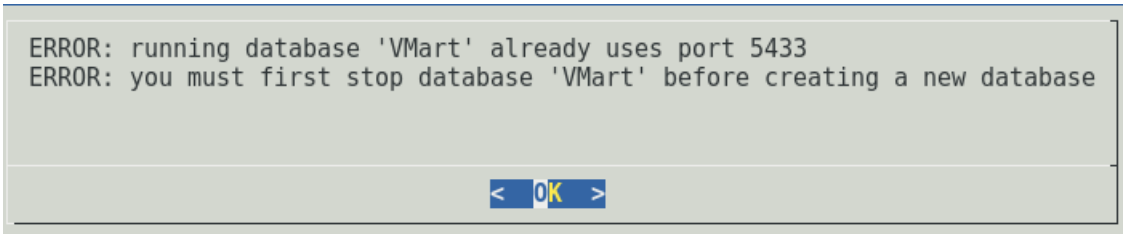


Рисунок 5.17 – Сообщение, если не была закрыта база данных VMart

В этом случае надо нажать кнопку «ОК», при этом происходит возврат к пункту «**Configuration Menu**» администратора базы данных. Надо будет закрыть демонстрационную базу данных «VMart» и повторить процесс создания новой базы данных.

Процесс создания базы данных начинается с выводом сообщений системы, как показано на рис. 5.18.

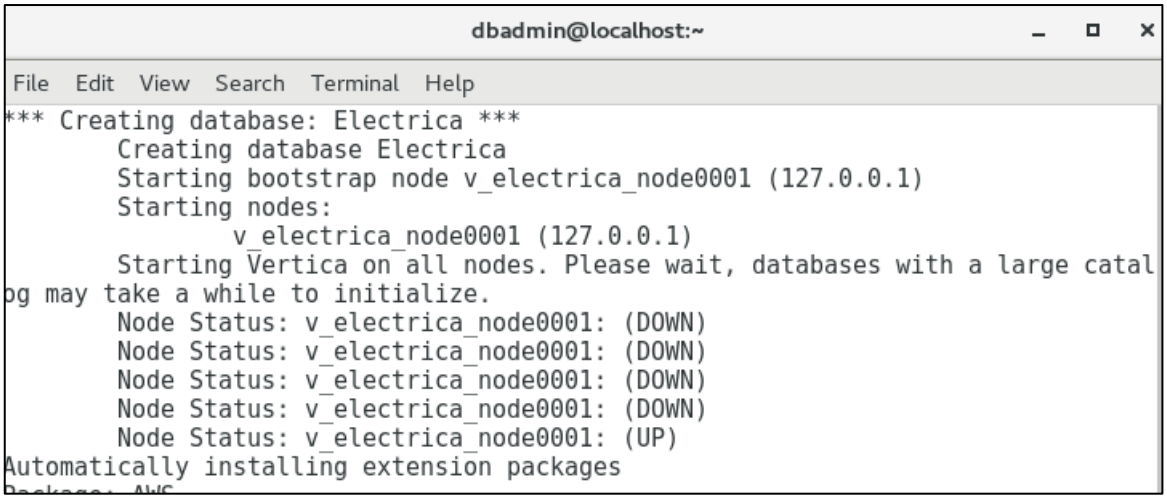


Рисунок 5.18 – Начало процесса создания БД

Примечание: создание БД в СУБД Vertica с использованием консоли – не очень быстрый процесс, в среднем эта операция занимает 5 минут. В течении этого времени не закрывайте консоль.

Спустя некоторое время в консоли появится сообщение с текстом «Database <name> created successfully» (рис. 5.19), что означает успешное завершение процесса создания БД.

Рисунок 5.19 – Сообщение об успешном создании БД

Теперь можно приступать к работе с базой данных. Как было уже сказано СУББД Vertica поддерживает язык SQL и снабжена стандартным SQL-интерфейсом (ANSI SQL-99), имеющим расширения для работы с аналитическими запросами. Поэтому для работы с данной СУБД достаточно знать стандартный язык SQL, который рассматривался в предыдущем практикуме. При нажатии кнопки **ОК** происходит переход к меню конфигурации, как показано на рис. 5.20.

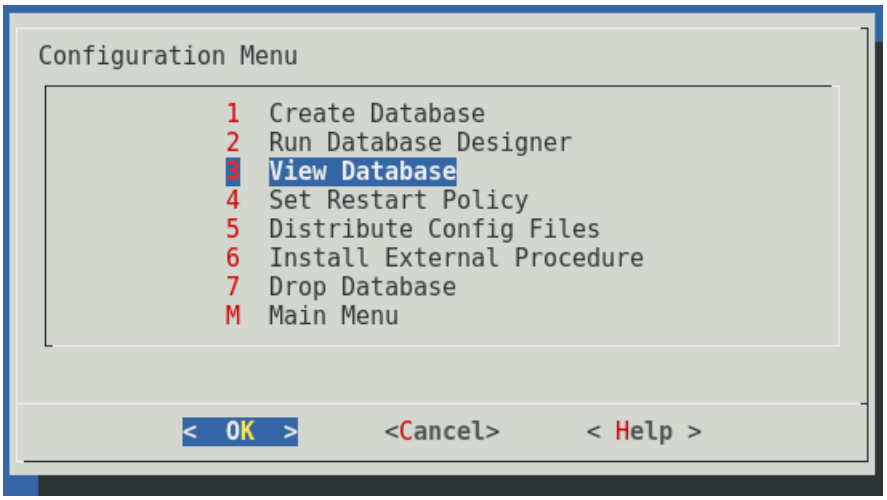


Рисунок 5.20 – Выбор пункта обзора базы данных

Для просмотра баз данных надо выбрать в меню конфигурации пункт «**View Database**». В результате будет открыто окно со списком баз данных, показанное на рис. 5.21.

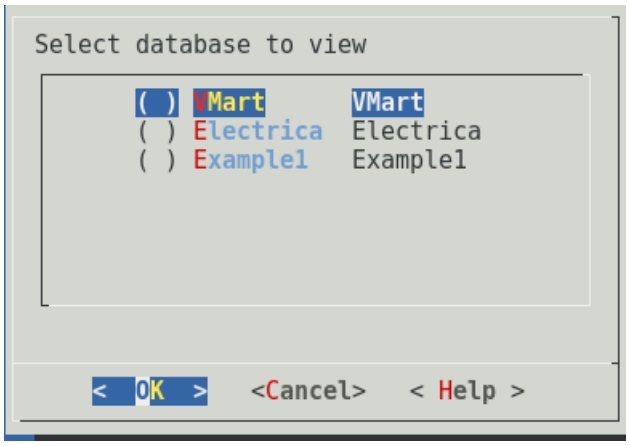


Рисунок 5.21 – Выбор базы данных

Выбрать из меню нужную базу данных. В результате появится сообщение, показанное на рис. 5.22, с информацией о базе данных.

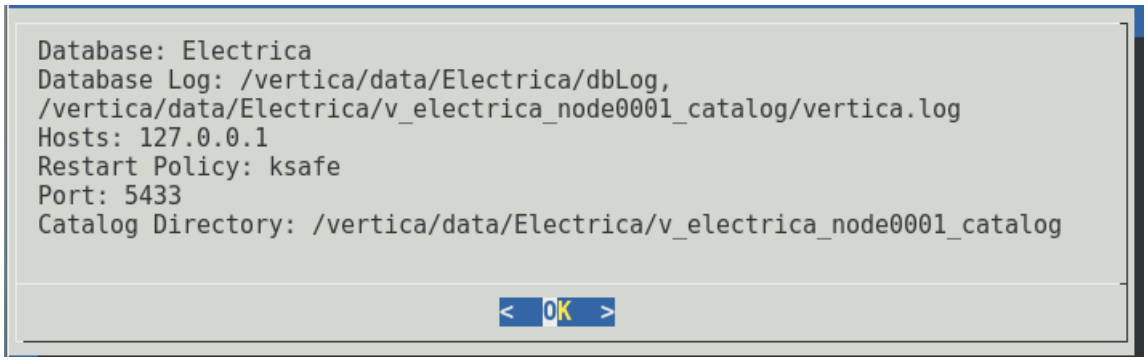


Рисунок 5.22 – Сообщение о параметрах выбранной базы данных

Для открытия созданной ранее БД выберем в главном меню пункт меню «**Start Database**», как показано на рис. 5.23. Пункт «**Main Menu**» возвращает управление в главное меню.

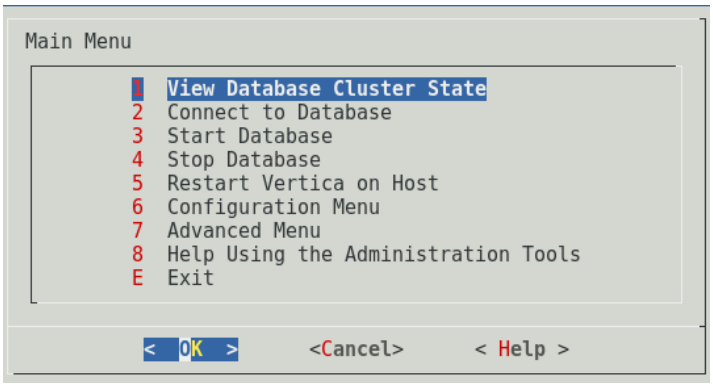


Рисунок 5.23 – Пункт меню для подключения к базе данных

После выбора данного пункта снова открывается окно терминала с командами системы (рис. 5.24), где можно приступить к работе с базой данных.

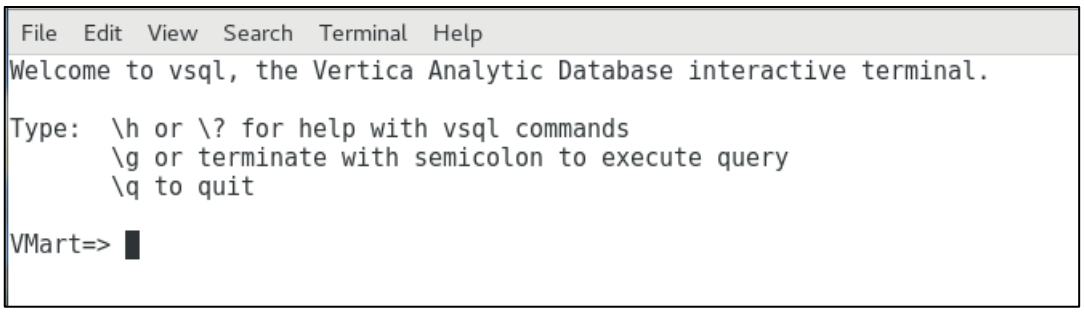


Рисунок 5.24 – Вид окна терминала после подключения к БД

Для примера рассмотрим создание таблицы, занесение в нее значений и их просмотр, как это выполнялось в предыдущих практикумах, но с графической консоли администратора.

Для того, чтобы создать таблицу, введем в терминале оператор SQL CREATE TABLE, как показано на рис. 5.25.

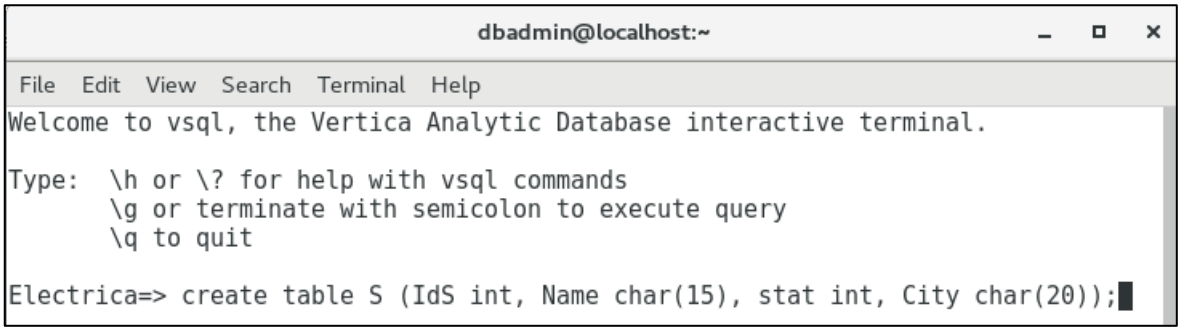


Рисунок 5.25 – Создание новой таблицы

Теперь введем в созданную таблицу данные. Добавим несколько строк при помощи оператора INSERT INTO.

Пример ввода операторов представлен на рис. 5.26.

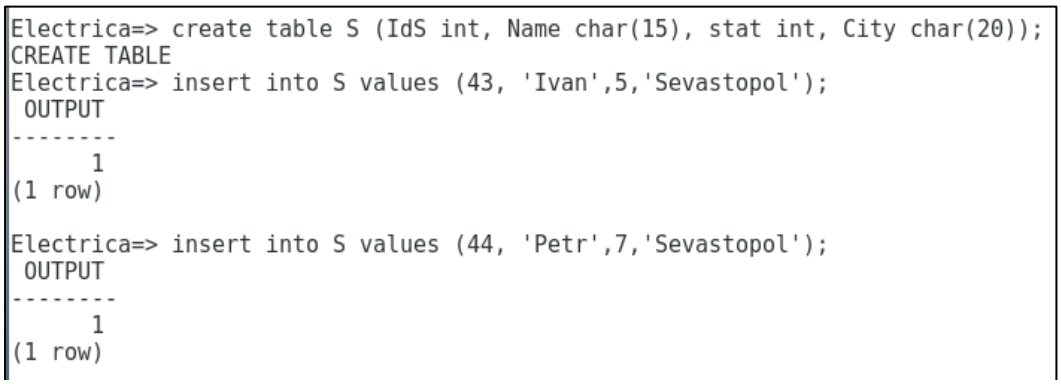


Рисунок 5.26 – Ввод значений

Обратите внимание, что после выполнения каждого оператора система выдает соответствующие сообщения. Например, CREATE TABLE (Создана таблица), OUTPUT 1 row (Создана одна строка).

Группу операций, которые были проведены с базой данных, необходимо подтвердить. Для этого выполнить в консоли оператор «COMMIT» (рис. 5.27). Только после выполнения этого оператора можно отсоединиться от базы данных, иначе можно потерять данные, которые были занесены в течении текущей транзакции.

```
Electrica=> insert into S values (45, 'Serg',8,'Simferopol');
 OUTPUT
--------
      1
(1 row)

Electrica=> commit;
COMMIT
Electrica=>
```

Рисунок 5.27 – Подтверждение операций, проводимых с БД

Теперь в базе содержится информация о трех записях. Для просмотра введенных данных необходимо воспользоваться оператором SELECT. Пример использования этого оператора показан на рис. 5.28.

```
Electrica=> select * from S;
 IdS |      Name       | stat |         City
-----+-----------------+------+--------------
  43 | Ivan            |    5 | Sevastopol
  44 | Petr            |    7 | Sevastopol
  45 | Serg            |    8 | Simferopol
(3 rows)

Electrica=>
```

Рисунок 5.28 – Вывод информации, содержащейся в таблице

Результат выдается в виде текстовых строк. Их можно скопировать в буфер и загрузить в любой файл основной машины.

Для перехода в главное меню системы надо в командной строке набрать последовательность символов «\q». Для выхода из базы данных остановить активную базу данных и в главном меню выбрать пункт “Exit”. При остановке системы появится окно, показанное на рис. 5.29. В списке активных баз данных надо поставить маркер в поле со скобками выбранной базы данных и нажать кнопку **ОК**.

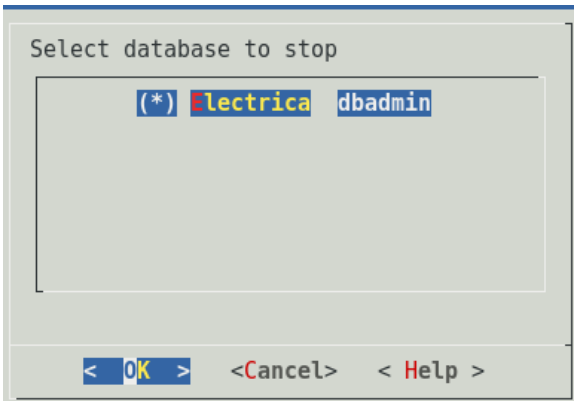


Рисунок 5.29 – Выбор базы данных для останова

Упражнение 5.1. Создать с помощью терминала новую базу данных с именем «**Example2**». В новой базе данных создать отношение «**Детали**» из предыдущих практикумов.

5.6 Пример построения регрессионной модели

Данный пример входит в список примеров аналитики, поставляемый вместе с системой. Для построения прогноза взяты данные об извержении знаменитого гейзера Олд Фейтфул, который находится в Йеллоустонском национальном парке, США. Извержения гейзера можно наблюдать и записывать. Данные и их статистический анализ можно посмотреть в интернете, например, в работе [12], где измерялось время между извержениями и продолжительность самого извержения. На рис. 5.30 показан график между двумя переменными, измеряемые в минутах. Wait Time – время покоя гейзера и Eruption Time – время извержения.

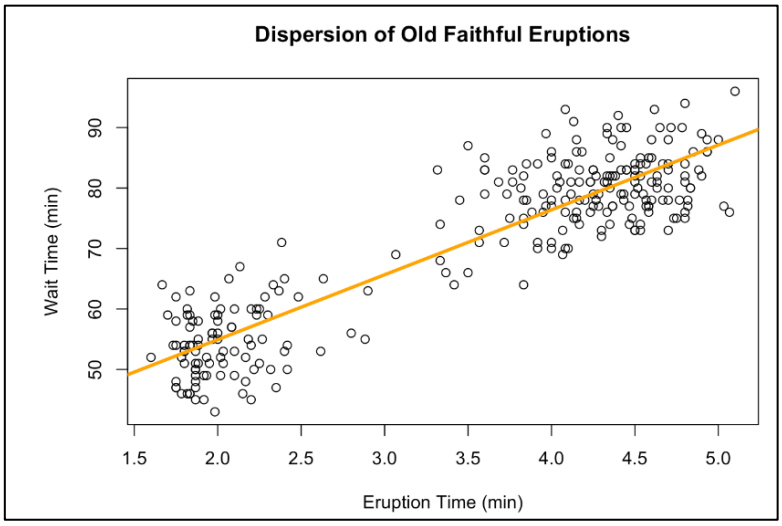


Рисунок 5.30 – Зависимость времени покоя гейзера
от продолжительности извержения

Требуется предсказать длительность следующего извержения. Исходные данные находятся в текстовом файле. На рис. 5.31 приведен пример первых шести строк файла.

```
"Ind","eruptions","waiting"
1,3.6,79
2,1.8,54
3,3.333,74
4,2.283,62
5,4.533,85
6,2.883,55
```

Рисунок 5.31 – Фрагмент текстового файла с исходными данными

Первая строка файла содержит три символьные константы с именами первых трех столбцов отношения, в которое будут загружаться данные. Остальные строки содержат номер строки с данными и собственно статистические данные. Данные разделяются запятыми. Файл должен быть подготовлен заранее. Как обмениваться файлами с главным компьютером описано в приложении А. Чтобы найти файл в системе Linux надо в главном меню выбрать пункт «**Places**», как показано на рис. 5.32.

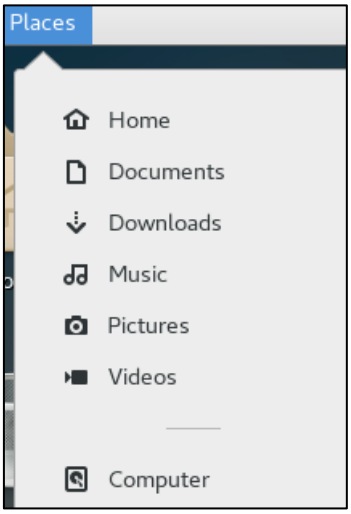


Рисунок 5.32 – Места хранения данных

Файл можно загрузить в папку пользователя с именем Home или в системную папку. В последнем случае надо выбрать пункт меню «**Computer**». В результате откроется окно с системными папками, как показано на рис. 5.33.

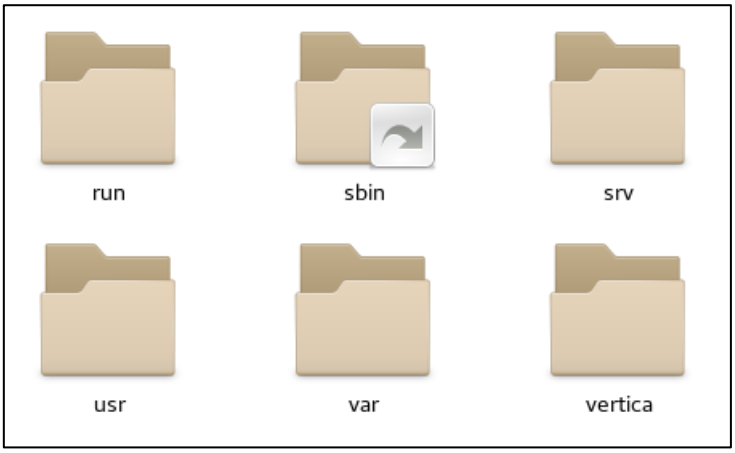


Рисунок 5.33 – Системные папки

В системной папке с именем «vertica» имеется только одна папка с именем «Data», а в ней три папки с данными для созданных баз, как показано на рис. 5.34.

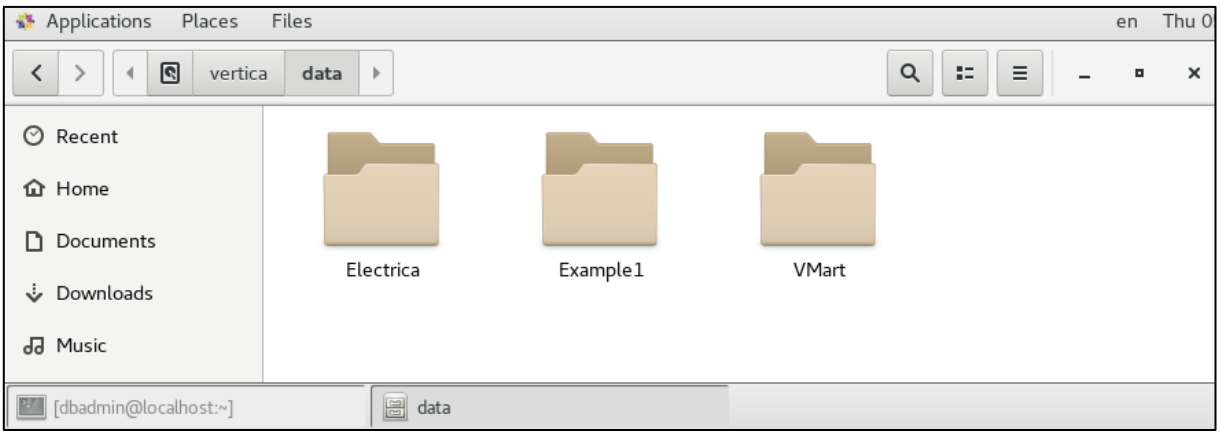


Рисунок 5.34 – Папки с данными для системы Vertica

Данные хранятся в отношении, созданном следующим оператором:

CREATE TABLE faithful (id int, eruptions float, waiting int);

Для расчета параметров линейной регрессионной модели первоначально войдем в управление базой данных с помощью команд консоли. Выполним в операторе SELECT функцию построения регрессионной модели:

=> SELECT LINEAR_REG('linear_reg_faithful', 'faithful', 'eruptions', 'waiting'
USING PARAMETERS optimizer='BFGS');

Здесь 'linear_reg_faithful' – имя создаваемой таблицы с результатами статистического анализа, 'faithful' – имя отношения с исходными данными,

'eruptions' – имя столбца с функцией отклика *y*, 'waiting' – имя столбца с фактором x. Для расчетов используется описанный выше метод оптимизации BFGS. Остальные параметры оператора используются по умолчанию.

В результате выполнения оператора будет выдано сообщение, показанное на рис. 5.35.

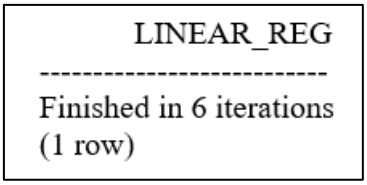


Рисунок 5.35 – Сообщение об успешном построении регрессионной модели

Обратите внимание, что никаких данных на выходе нет. Все параметры модели записываются в специальном отношении, в котором имеется атрибут – «Имя модели». В нашем случае имя модели - linear_reg_faithful.

Для вывода в нужном виде результатов статистического анализа модели linear_reg_faithful используется специальная функция, как показано ниже.

```
=> SELECT GET_MODEL_SUMMARY(USING PARAMETERS
model_name='linear_reg_faithful');
```

В результате будут выданы данные, показанные на рис. 5.36.

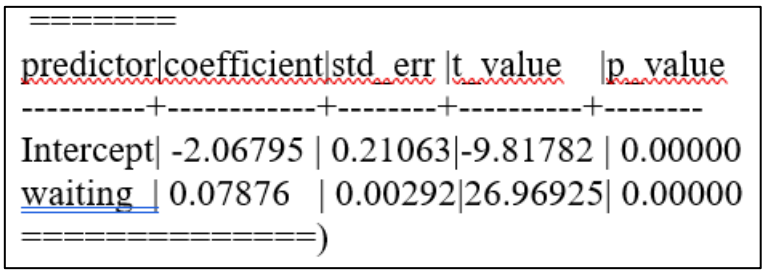


Рисунок 5.36 – Результаты регрессионного анализа

В таблице результатов отображаются значения следующих показателей модели: predictor – оцениваемый фактор; coefficient – оценка коэффициента модели α_i; std_err – среднее квадратическое отклонение оценки коэффициента; t_value – значение критерия Стьюдента; p_value – значение вероятности, соответствующее вычисленному значению критерия; Intercept – свободный коэффициент уравнения регрессии.

Для вычисления прогнозного значения по полученной модели используется функция PREDICT_LINEAR_REG. Предварительно для нее

надо создать новую таблицу на основе запроса к существующей таблице с результатами регрессионного анализа. Синтаксис этой функции следующий:

PREDICT_LINEAR_REG (input-columns USING PARAMETERS model_name='model-name' [, match_by_pos=match-by-position]).

Здесь input-columns - список атрибутов исходного отношения через запятую или звездочка (*) для выбора всех атрибутов; model_name - имя модели (заглавные и строчные буквы не различаются case-insensitive); match_by_pos - булевская переменная, которая определяет способ, каким атрибуты вводятся в модель (Значения: false (по умолчанию): по именам. true: по месту атрибута в списке).

Предсказанные значения и значения факторов должны быть в одной таблице. Для создания этой таблицы и загрузки в нее результатов расчетов можно воспользоваться оператором создания таблицы по подзапросу.

В рассматриваемом примере имя таблицы для прогноза pred_faithful_results. Оператор создания таблицы с результатами прогноза имеет следующий вид:

=> CREATE TABLE pred_faithful_results AS (SELECT id, eruptions, PREDICT_LINEAR_REG(waiting USING PARAMETERS model_name='linear_reg_faithful') AS pred FROM faithful_testing);

В результате выполнения оператора выводится сообщение CREATE TABLE, при этом результирующая таблица не отображается.

Для просмотра результата расчетов надо выполнить следующий оператор:

=> SELECT * FROM pred_faithful_results ORDER BY id;

В результате будет выдана таблица, фрагмент которой показан на рис. 5.37.

id	eruptions	pred
4	2.283	2.8151271587036
5	4.533	4.62659045686076
8	3.6	4.62659045686076

Рисунок 5.37 – Фрагмент таблицы с результатами расчета прогнозных значений

Для дальнейшей обработки полученных прогнозных данных в системе существует специальная функция MSE вычисления среднего всех квадратов

отклонений реальных значений функции отклика от предсказанных значений. Синтаксис этой функции следующий:

MSE (*targets*, *predictions*) OVER()

Параметры функции следующие: *targets* - столбец с реальными значениями выходной переменной (тип: FLOAT); *predictions* - столбец, содержащий спрогнозированные значения для выходной переменной (тип: FLOAT).

В рассматриваемом примере оператор вычисления остаточной суммы квадратов имеет следующий вид:

```
=> SELECT MSE (eruptions::float, pred::float) OVER() FROM
(SELECT eruptions, pred FROM pred_faithful_results) AS prediction_output;
```

В результате выполнения оператора будет выдана информация о средней остаточной сумме квадратов, показанная на рис. 5.38.

```
mse                 | Comments
--------------------+-------------------------------------------
0.252925741352641 | Of 110 rows, 110 were used and 0 were ignored
(1 row)
```

Рисунок 5.38 – Расчет средней остаточной суммы квадратов

В системе имеется еще несколько полезных функций для расчета статистических параметров полученной регрессионной модели.

Упражнение 5.2. В следующей таблице показаны условные результаты эксперимента.

X1	X2	Y
1	3	5
1	3	6
1	4	7
1	4	10
2	3	12
2	3	18
2	4	16
2	4	30

В таблице X1 и X2 – входные переменные, Y – выходная переменная.

Требуется выполнить следующие шаги для выполнения аналитических расчетов.

Шаг 1. Создать отношение для экспериментальных данных. Оператор для выполнения этого действия показан на рис. 5.39.

```
1 create table Data1 (x1 float, x2 float, y float);
```

Рисунок 5.39 – Оператор создания новой таблицы

Действия с отношениями можно выполнять в консоли менеджера базы данных.

Шаг 2. Загрузить экспериментальные данные в таблицу. Операторы для выполнения этого действия показан на рис. 5.40.

```
1 insert into Data1 values(1,3,5);
2 insert into Data1 values(1,3,6);
3 insert into Data1 values(1,4,7);
4 insert into Data1 values(1,4,10);
5 insert into Data1 values(2,3,12);
6 insert into Data1 values(2,3,18);
7 insert into Data1 values(2,4,16);
8 insert into Data1 values(2,4,30);
9 commit;
```

Рисунок 5.40 – Операторы загрузки данных

Шаг 3. Проверить загруженные данные с помощью оператора SELECT, как показано на рис. 5.41.

```
1 select * from Data1;
2
```

Рисунок 5.41 – Оператор для проверки данных

Результат выполнения оператора показан на рис. 5.42.

```
VMart=> select * from Data1;
 x1 | x2 | y
----+----+----
  1 |  3 |  5
  1 |  3 |  6
  1 |  4 |  7
  1 |  4 | 10
  2 |  3 | 12
  2 |  3 | 18
  2 |  4 | 16
  2 |  4 | 30
(8 rows)
```

Рисунок 5.42 – Результат проверки данных

Шаг 4. Построить двухфакторную линейную регрессионную модель с помощью оператора показанного на рис. 5.43.

```
VMart=> select linear_reg('Model1','Data1','y','x1,x2' using parameters optimizer='BFGS');
```

Рисунок 5.43 – Оператор построения регрессионной модели

Этот оператор выполняется с командного терминала операционной системы. Результат выполнения оператора показан на рис. 5.44.

```
        linear_reg
--------------------------
 Finished in 4 iterations

(1 row)
```

Рисунок 5.44 – Сообщение об успешном выполнении оператора

Шаг 5. Вывести параметры полученной модели с помощью оператора, показанного на рис. 5.45.

```
VMart=> select get_model_summary(using parameters model_name='Model2');
```

Рисунок 5.45 – Оператор вывода результатов моделирования

Результаты работы оператора показаны на рис. 5.46.

```
get_model_summary

=======
details
=======
predictor|coefficient|std_err |t_value |p_value
---------+-----------+--------+--------+--------
Intercept| -24.25000 |14.03255|-1.72812| 0.14454
   x1    | 12.00000  | 3.65377| 3.28428| 0.02185
   x2    |  5.50000  | 3.65377| 1.50530| 0.19259
```

Рисунок 5.46 – Результаты моделирования

На рис. 5.47 даны дополнительные более подробные результаты селекции моделей.

```
===============
regularization
===============
type| lambda
----+--------
none| 1.00000


============
call_string
============
linear_reg('public.Model2', 'Data1', '"y"', 'x1,x2'
USING PARAMETERS optimizer='bfgs', epsilon=1e-06, max_iterations=10
0, regularization='none', lambda=1, alpha=0.5)

================
Additional Info
================
       Name       |Value
------------------+-----
 iteration_count  |  4
rejected_row_count|  0
accepted_row_count|  8

(1 row)
```

Рисунок 5.47 – Параметры селекции полученной модели

Шаг 6. Разобраться в распечатке и дать оценку полученным результатам. В рассматриваемом примере для уровня значимости 0.05 по критерию Стьюдента свободный коэффициент и второй коэффициент незначимы, так как вероятность P для них больше 0.05.

5.7 Порядок выполнения работы

Шаг 1. Загрузить виртуальную машину с системой.

Шаг 2. Выполнить упражнени5.1.

Шаг 3. Заменить в таблице данные согласно индивидуальному варианту. Столбец Y заполняется согласно варианту, а столбцы X1 и X2 во всех вариантах остаются одинаковыми.

Шаг 4. Построить регрессионную модель вида

$$y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \varepsilon.$$

Коэффициенты модели и данные статистического анализа модели получить с помощью функции

GET_MODEL_SUMMARY(USING PARAMETERS model_name='Имя Вашей модели');

Имя модели при выполнении выбирает сам студент.

Шаг 5. Дать прогноз значения функции отклика Y в точке X1 = 3 и X2 = 5. Для этого добавить в таблицу Data1 дополнительную строку, как показано на рис. 5.48.

```
VMart=> insert into Data1 values (3,5);
 OUTPUT
--------
      1
(1 row)
```

Рисунок 5.48 – Добавление строки для прогноза

Не забыть закончить транзакцию, как показано на рис. 5.49.

```
VMart=> commit;
COMMIT
```

Рисунок 5.49 – Окончание транзакции

Получить прогнозное значение с помощью оператора, показанного на рис. 5.50.

```
VMart=> create table pred_v1 as (select x1,x2,predict_linear_reg(x1,x2
using parameters model_name='Model2') as prediction from Data1);
```

Рисунок 5.50 – Вычисление прогноза

Просмотреть результаты прогноза. Для этого воспользоваться оператором SELECT * FROM pred_v1; Результат может иметь вид, показанный на рис. 5.51.

```
 x1 | x2 |    prediction
----+----+------------------
  1 |  3 | 4.25000014222951
  1 |  3 | 4.25000014222951
  1 |  4 |  9.7500001632867
  1 |  4 |  9.7500001632867
  2 |  3 | 16.2500001455807
  2 |  3 | 16.2500001455807
  2 |  4 | 21.7500001666379
  2 |  4 | 21.7500001666379
  3 |  5 | 39.2500001910463
(9 rows)
```

Рисунок 5.51 – Прогнозные значения

Здесь получены значения прогноза для всех точек в таблице. Если нас интересует только точка прогноза, то запрос можно уточнить. Либо в операторе создания новой таблицы, как показано на рис. 5.52.

```
VMart=> create table pred_v2 as (select x1,x2,predict_linear_reg(x1,x2
using parameters model_name='Model2') as prediction from Data1 where x1
=3 and x2=5);
```

Рисунок 5.52 – Прогноз в одной точке

Либо в запросе на результат прогноза в таблице pred_v1, как показано на рис. 5.53.

```
VMart=> select * from pred_v1 where x1=3 and x2=5;
 x1 | x2 |    prediction
----+----+-----------------
  3 |  5 | 39.2500001910463
(1 row)
```

Рисунок 5.51 – Выбор одной точки

Шаг 6. Оценить ошибку прогноза в заданной точке с помощью функции MSE.

5.8 Содержание отчета

Отчет должен содержать следующую информацию: копии окон экрана с выполненным заданием, выводы.

5.9 Контрольные вопросы

1. Что такое регрессионный анализ и для чего он используется?
2. Почему не используется стандартный метод наименьших квадратов?
3. Что такое критерий Стьюдента?
4. Что такое критерий Фишера?
5. Что такое ошибка прогноза?
6. Что такое метод регуляризации?
7. Какими возможностями обладает программная система Vertica в области аналитики?

5.10 Варианты задания

В следующей таблице заданы значения переменной Y для вариантов задания.

Варианты													
1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	1	7	3	2	1	2	3	3	10	3	1	2	5
8	2	7	4	2	2	4	4	7	9	3	1	1	4
7	3	7	3	2	3	5	5	2	8	7	1	2	7
5	2	8	4	3	3	2	6	4	8	4	3	7	5
7	2	9	5	4	3	2	6	6	4	8	6	9	6
2	4	1	6	5	4	6	7	8	4	3	3	9	6
4	4	10	7	7	5	8	7	8	1	6	9	9	8
3	5	9	7	7	6	9	9	9	2	9	9	6	9

Список литературы

1. О Стратегии развития информационного общества в Российской Федерации на 2017 - 2030 годы : Указ Президента РФ от 9 мая 2017 г. N 203. — Текст : электронный // КонсультантПлюс : справ. правовая система. — 1992 — . — URL: https://login.consultant.ru/link/?req=doc&base=LAW&n=216363&dst=100008 (дата обращения: 10.01.2025). — Режим доступа: из локальной сети Севастопольского государственного университета.

2. Губанов Д. А. Социальные сети: модели информационного влияния, управления и противоборства / Губанов Д. А., Новиков Д. А., Чхартишвили А. Г. ; под ред. чл.-корр. РАН Д. А. Новикова.— М. : Изд-во физико-математической литературы, 2010.—228 с.

3. Гула Е. Большие данные BigData для HR . Как увидеть личность за цифрой? / Гула Е., Канардов И. — Текст : электронный // HR-MEDIA.RU : Журнал компетенции. Об управлении персоналом, карьере и эффективности сотрудников. — URL: http://hr-media.ru/bolshie-dannye-bigdata-dlya-hr-kak-uvidet-lichnost-za-tsifroj/ (дата обращения 10.01.25).

4. Дейт К. Руководство по реляционной СУБД DB2. — М. : Финансы и статистика, 1988.— 320 с. — ISBN 5—279—00063—9.

5. Савкин К. С. Как использовать BigDate в бизнесе / Савкин К. С. — Текст : электронный. — URL: https://www.savkinks.ru/bigdate-business.htm (дата обращения 10.01.25).

6. Большие данные в России: что изменилось для бизнеса и государства / сост. А. Погорельский. — Текст : электронный // РБК Тренды. — URL: https://trends.rbc.ru/trends/industry/cmrm/64e607689a7947cb6fb21213 (дата обращения 10.01.25).

7. Большая гонка за большими данными / сост. Киселев Д. — Текст : электронный // Buying Business Travel Russia: Портал для профессионалов в области бизнес-туризма и MICE. — URL: https://buyingbusinesstravel.com.ru/articles/stati/bolshaya-gonka-za-bolshimi-dannymi (дата обращения 10.01.25).

8. Работа с Big Data: основные области и возможности / по материалам research&trends. — Текст : электронный // Энциклопедия маркетинга. — URL: http://www.marketing.spb.ru/lib-around/stat/Big_Data.htm (дата обращения 10.01.25).

9. Старт big data проекта: 6 важных вопросов / сост. Гришина А. // Хабр. New Professions Lab. — URL: https://habr.com/ru/companies/newprolab/articles/329544/ (дата обращения 10.01.25).

10. Профессиональные услуги в сфере BI. Big Data и Vertica: что это, зачем и для кого? // АналитикаПлюс. — URL: https://analytikaplus.ru/big-data-i-vertica-chto-eto-zachem-i-dlya-kogo/ (дата обращения 10.01.25).

11. Oracle VM VirtualBox // ORACLE. — URL: https://www.oracle.com/virtualization/virtualbox/index.html (дата обращения 10.01.25)

12. Debyn Schutz Patterns of Eruption: Old Faithful / Debyn Schutz. — Текст : электронный // RПабы. — URL: https://rpubs.com/schutzd1/249593 (дата обращения 10.01.25).

Приложение А

Установка и настройка системы VIRTUALBOX

А.1 Установка VirtualBox

VirtualBox (Oracle VM VirtualBox) – это кросс-платформенное средство программной виртуализации для операционных систем Microsoft Windows, Linux, FreeBSD, Mac OS X, Solaris/OpenSolaris, ReactOS, DOS и других.

Установка программы выполняется стандартными способами для используемой ОС и не вызывает затруднений. Для начала работ по установке продукт VirtualBox необходимо скачать с официального сайта (virtualbox.org).

На главной странице сайта необходимо нажать кнопку, показанную на рис. А.1.

Рисунок А.1 – Официальная страница загрузки VirtualBox

После выполнения данного действия откроется страница, где будет предложено выбрать соответствующую операционную систему (рис. А.2).

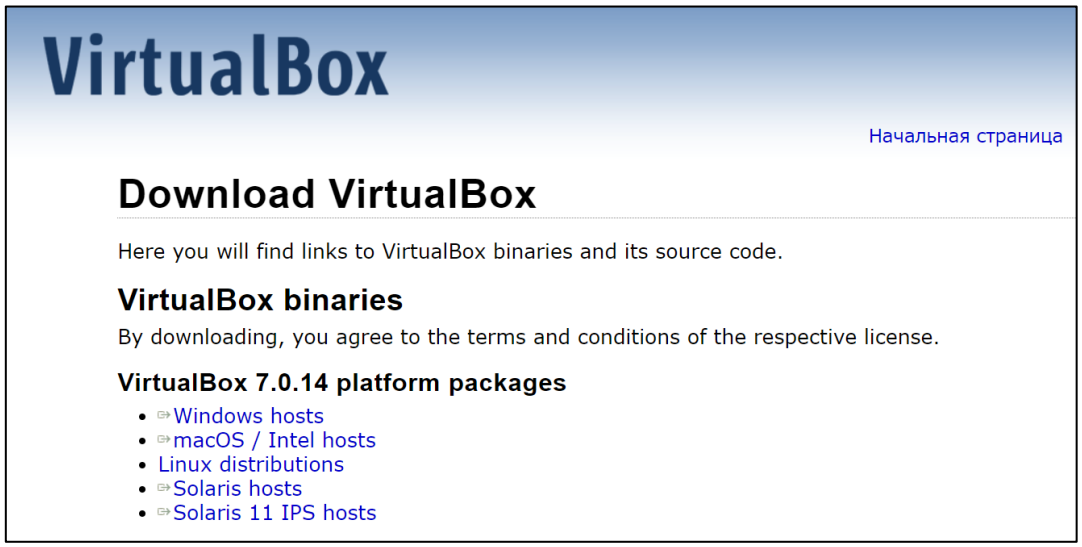


Рисунок А.2 – Выбор необходимой операционной системы

В данной инструкции рассматривается пример установки VM для ОС Windows (т.е. для скачивания выбран пакет «Windows hosts»).

Нажатие на доступный для скачивания пакет приведет к загрузке установочного файла (рис.А.3), который необходимо открыть по ее завершении.

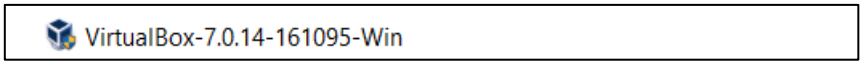


Рисунок А.3 – Пример загрузочного файла

Установка производиться стандартным образом (никаких дополнительных настроек производить не надо): после нескольких нажатий кнопки «Next», начнется непродолжительный процесс установки VirtualBox на Вашей машине. При установке VirtualBox на Windows 10 должны появляться окна «Безопасность Windows». На всех вопросах жмем «Установить». Можно активировать опцию «Всегда доверять программному обеспечению Oracle», чтобы минимизировать число таких окон.

После установки желательно перезагрузить компьютер и затем уже приступать к работе с данным продуктом.

А.2 Импорт виртуальной машины Vertica (VM)

Для того, чтобы приступить к работе с СУБД Vertica, необходимо импортировать конфигурацию уже созданной виртуальной машины из файла с именем «vertica_community_edition-9.1.1.ova».

Для начала импорта нажмите в окне OracleVM выберите в меню «Файлы» пункт «Импорт конфигурации» (рис. 6.4). В открывшемся окне выберите файл с архивом виртуальной машины. Файл при этом должен быть расположен на любом доступном операционной системе носителе.

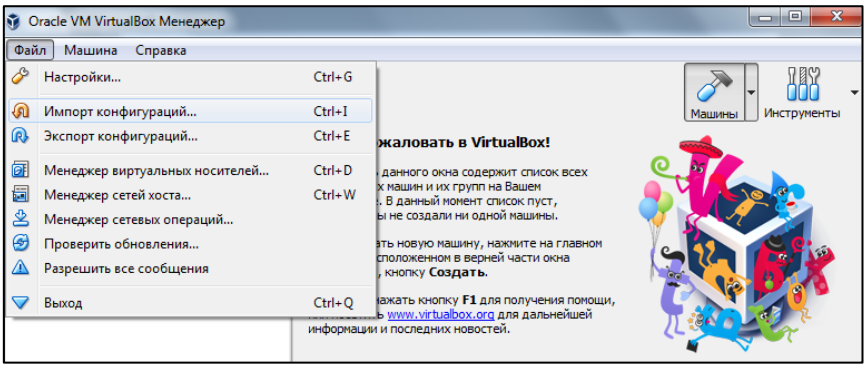


Рисунок А.4 – Выбор пункта «Импорт конфигураций»

После нажатия кнопки «Далее», появляется окно, показанное на рис. А.5.

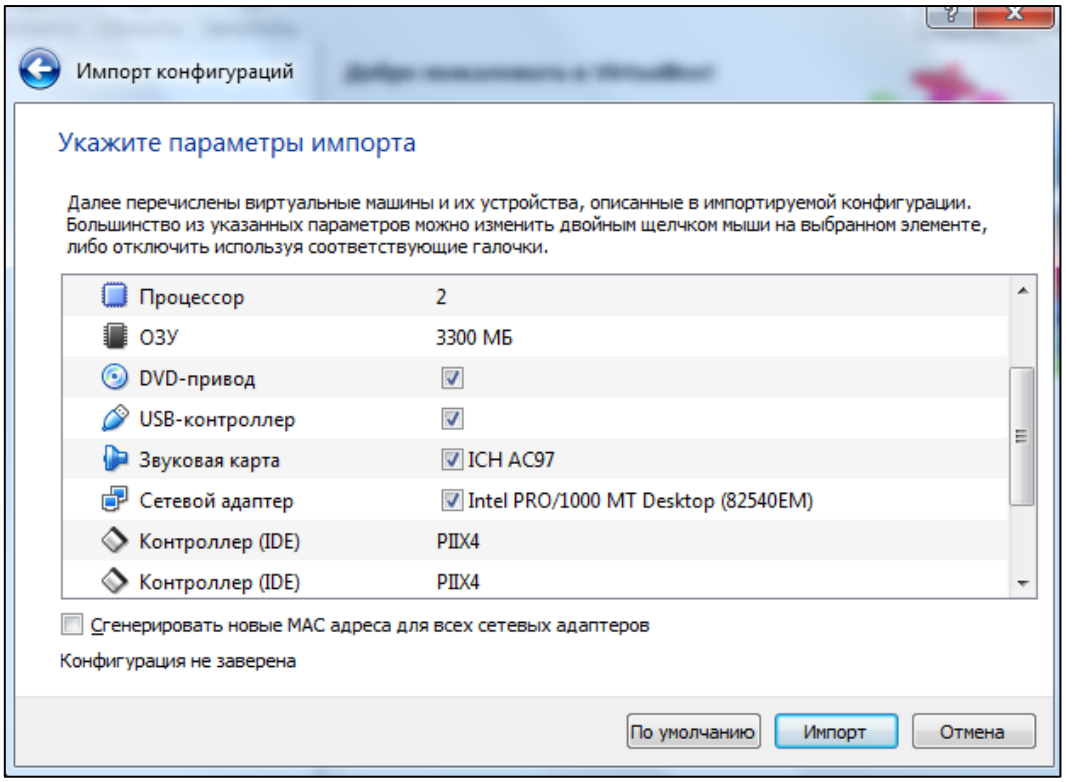


Рисунок А.5 – Выбор параметров импорта

Тут важным моментом является согласование объёма оперативной памяти импортируемой машины и компьютера. VirtualBox позволяет использовать во время работы ВМ максимум половину оперативной памяти компьютера. ВМ Vetrica настроена на использование около 4 Гб. оперативной памяти. Если на вашей компьютере недостаточно оперативной памяти для работы с такими параметрами конфигурации, то необходимо будет снизить это значение.

Нажимаем кнопку «Импорт» и дожидаемся завершения данной операции. Примечание: процедура импорта (рис. А.6) занимает достаточно большое время (от 4 до 20 минут).

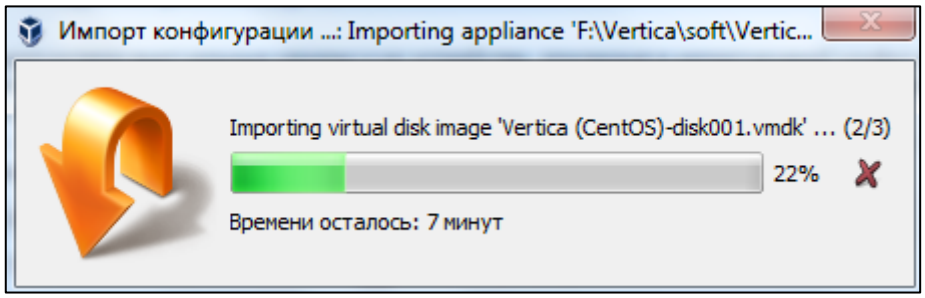


Рисунок А.6 – Процедура импорта

После успешного импорта, в окне доступных виртуальных машин должна появиться иконка ВМ Vertica, как это показано на рис. А.7 (В данном примере имя машины было Vertica (CentOS)).

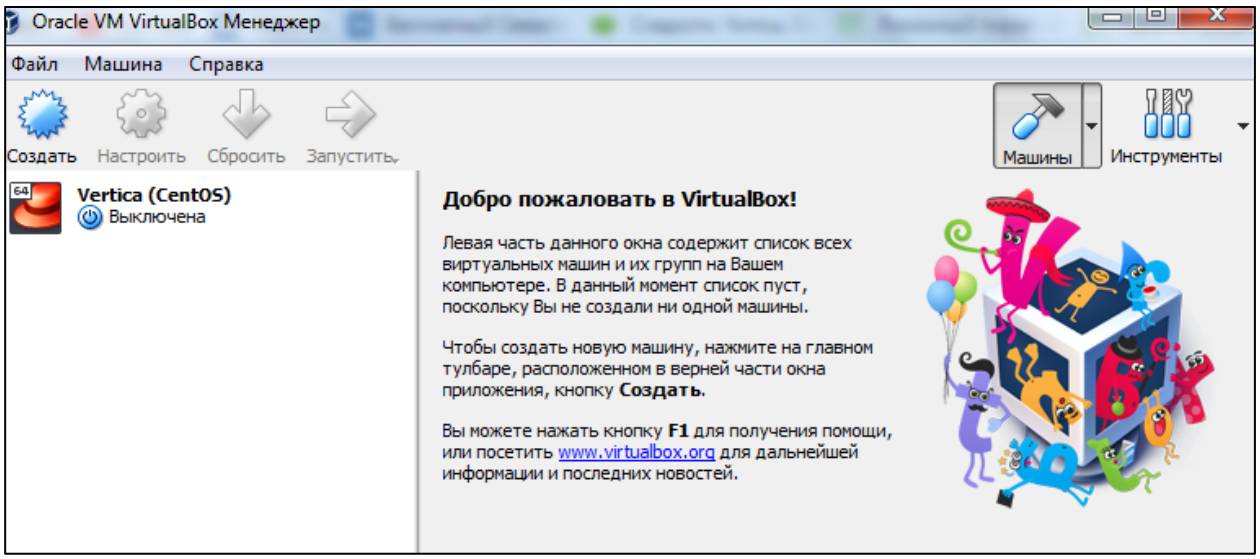


Рисунок А.7 – Вид окна доступных виртуальных машин

А.3 Настройка виртуальной машины

Перед запуском виртуальной машины, необходимо провести некоторые изменения в параметрах.

Нажимаем правую кнопку мыши (ПКМ) в меню для нашей виртуальной машины и выбираем пункт «Настроить…» (рис. А.8).

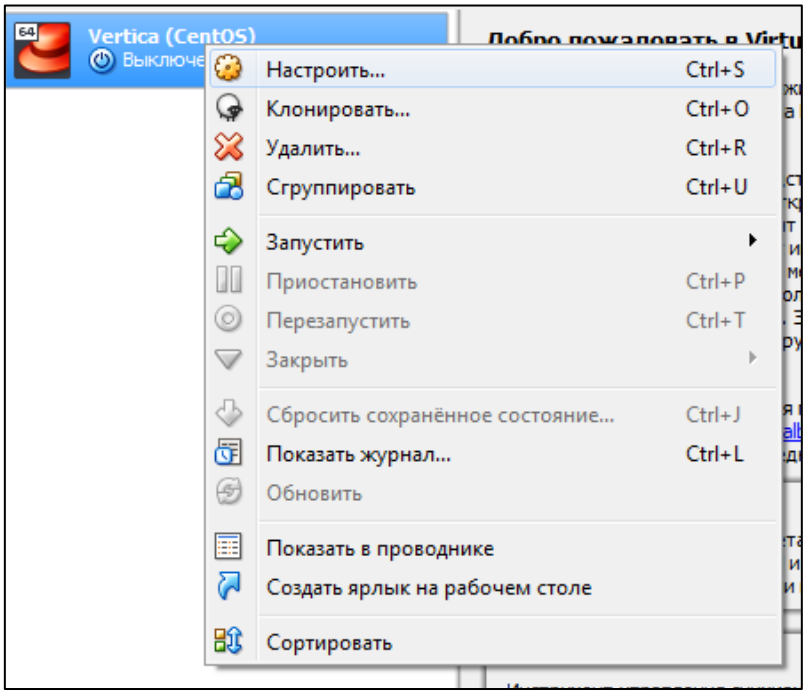


Рисунок А.8 – Выбор пункта «Настроить…»

Первое, на что следует обратить свое внимание, как это было написано выше, – объём оперативной памяти. Если его значение превысит половину оперативной памяти компьютера, то VirtualBox не сможет позволить Вам работать с соответствующей ВМ.

Пример настройки виртуальной машины в данных методических указаниях выполняется на компьютере с 4 ГБ оперативной памяти. В связи с этим снизим объем оперативной памяти, который будет выделен виртуальной машине.

Для этого выберем пункт «Система» в настройках и передвинем бегунок так, чтобы параметр «Основная память» принял значение 1024 МБ (т.е. ВМ получит 1 ГБ памяти), как это показано на рис. А.9. Если объем оперативной память позволяет, то этот пункт делать не надо.

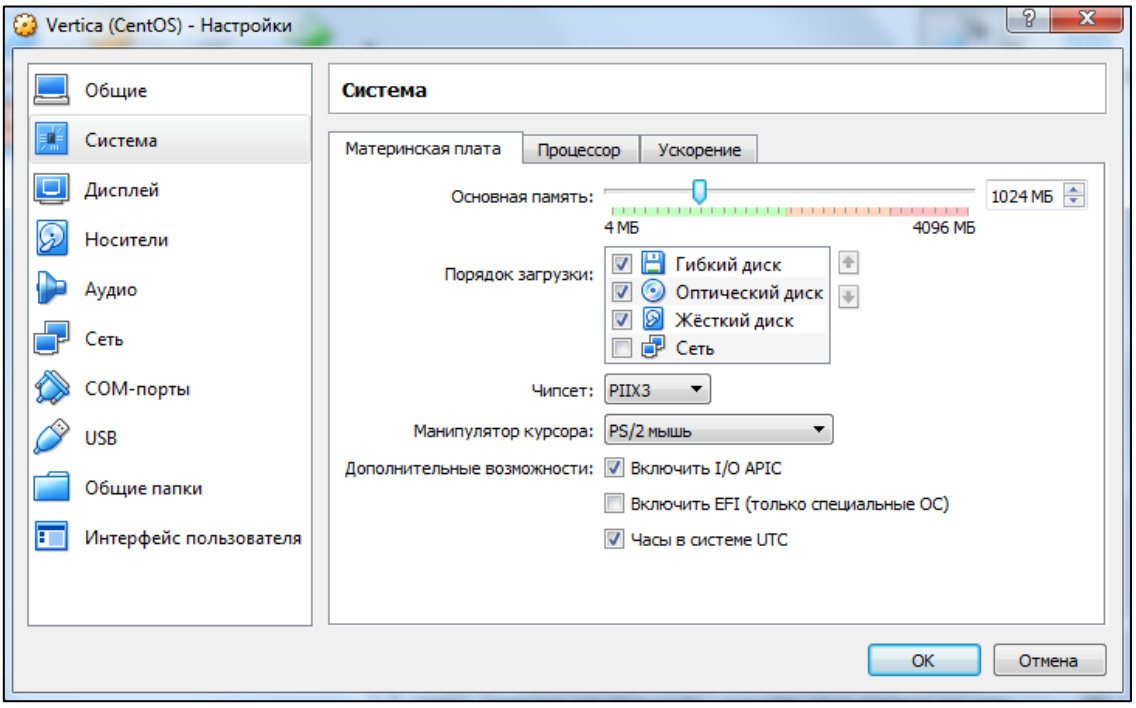


Рисунок А.9 – Изменение параметра «Основная память»
в настройках виртуальной машины

Второе, что необходимо изменить в настройках, касается параметра «Сеть». В случае импорта машины, созданной на другом компьютере, могут не совпадать адаптеры сети. Надо выбрать соответствующий сетевой адаптер реальной машины.

После выбора пункта «Сеть» откроется окно, показанное на рис. А.10. Используемый базовой машиной сетевой мост можно найти в списке, который открывается галочкой.

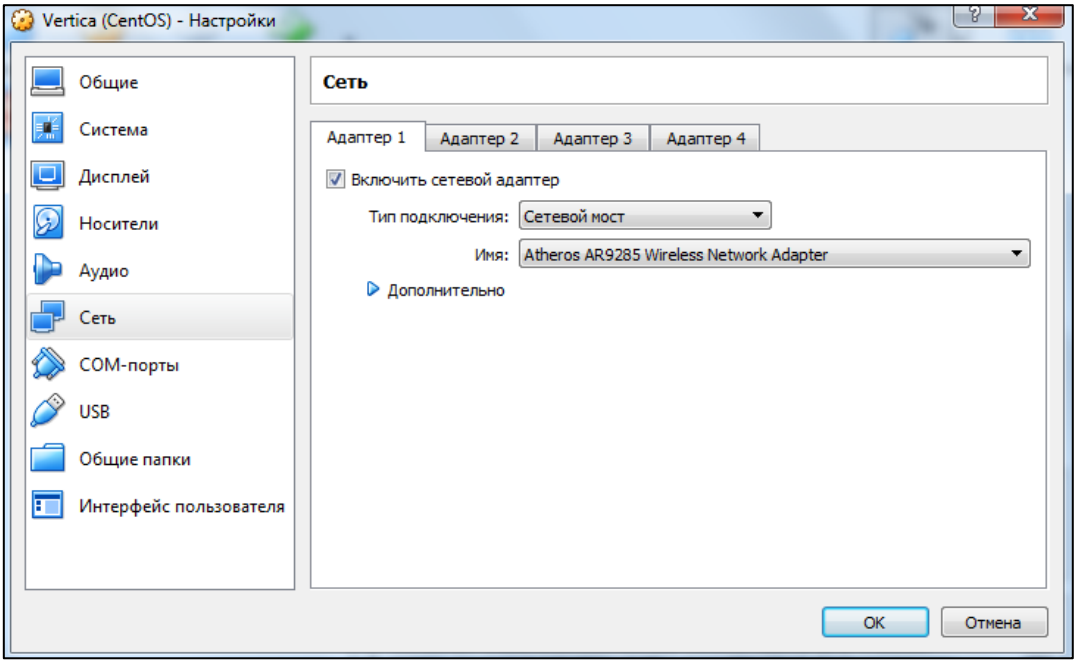


Рисунок А.10 – Настройки сети ВМ

Тут нажимаем на пункт «Дополнительно», где изменяем параметр «Неразборчивый режим» на «Разрешить все», как это показано на рис. А.11.

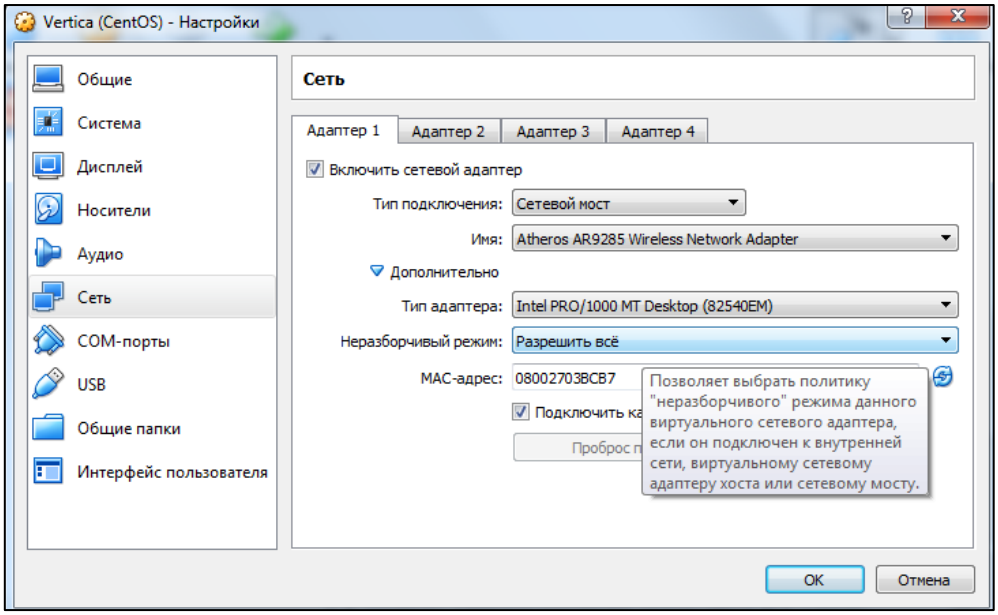


Рисунок А.11 – Изменение параметра «Неразборчивый режим»

Сохраняем изменения посредством нажатия «ОК». Теперь можно приступать к работе с ВМ, нажав кнопку «Запустить» в окне выбора виртуальных машин (рис. А.12).

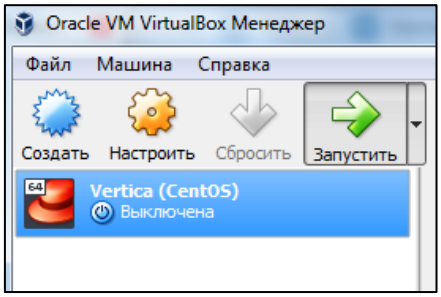


Рисунок А.12 – Запуск ВМ Vertica (CentOS)

Вид окна при запуске виртуальной машины представлен на рис. А.13.

Рисунок А.13 – Вид окна при запуске виртуальной машины

А.4 Обмен файлами между реальной и виртуальной операционными системами

При одновременной работе в основной и виртуальной операционной системе возникает необходимость переноса различных файлов из одной системы в другую. Осуществить это в VirtualBox можно двумя способами:

посредством создания «общей папки», которая будет доступна для обмена файлами между обеими операционными системами;

посредством использования флэш-накопителя.

В настоящем разделе рассматривается случай обмена данными между двумя машинами с помощью общей папки. Первоначально рассматривается случай, когда основная и виртуальная машины используют операционную систему Windows, а затем рассматриваются особенности для виртуальной машины с операционной системой Linux.

А.5 Создание общей папки в основной машине

Пример создания общих папок выполняется на ПК, основная ОС которого – Windows.

Для начала необходимо создать папку (которая в дальнейшем будет рассматриваться, как «общая») в основной операционной системе.

К примеру, создадим папку «Public» на локальном диске D (рис. А.14).

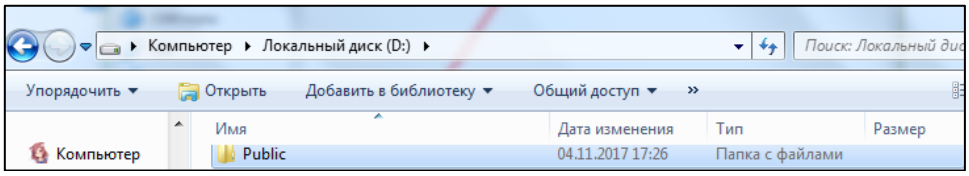


Рисунок А.14 – Создание папки в основной ОС

К папке должен быть открыт общий доступ. Для этого надо открыть пункт свойств папки, а затем открыть общий доступ к папке, как показано на вкладке «Доступ», как показано на рис. А.15.

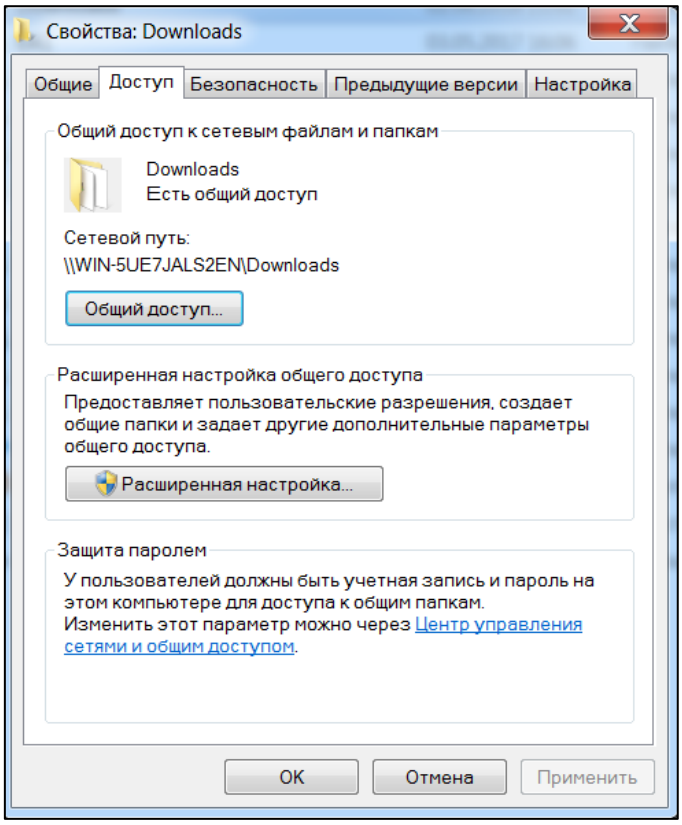


Рисунок А.15 – Окно для открытия общего доступа к папке

А.6 Создание образа папки в виртуальной машине в ОС Windows

Для создания образа общей папки в виртуальной машине необходимо открыть VirtulBox и найти в списке доступных виртуальных машин ту, для которой необходимо настроить общие папки. Нажать на иконку правой кнопкой мыши (далее ЛКМ – левая кнопка мыши, ПКМ – правая кнопка мыши) и выбрать пункт меню «Настроить…» (рис. А.16).

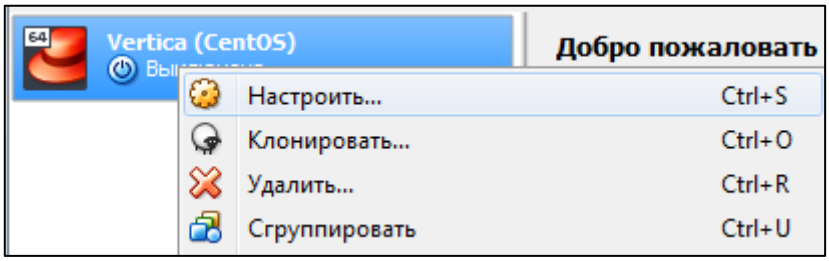


Рисунок А.16 – Пункт «Настроить…»

Затем в открывшемся окне настроек выбрать пункт меню «Общие папки» (рис. А.17).

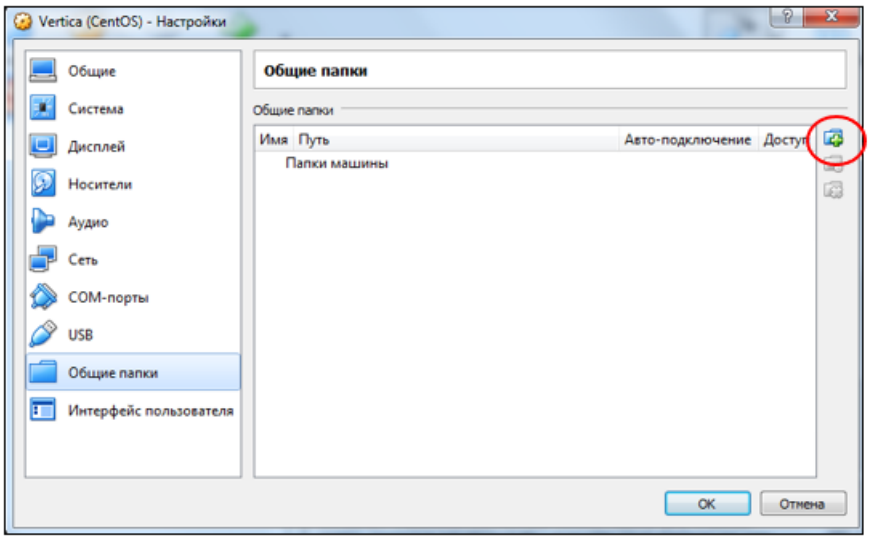


Рисунок А.17 – Пункт меню настроек «Общие папки»

Нажать на значок добавления папки, который выделен на рис. А.17 красным кружком. После этого действия должно открыться окно, показанное на рис. А.18.

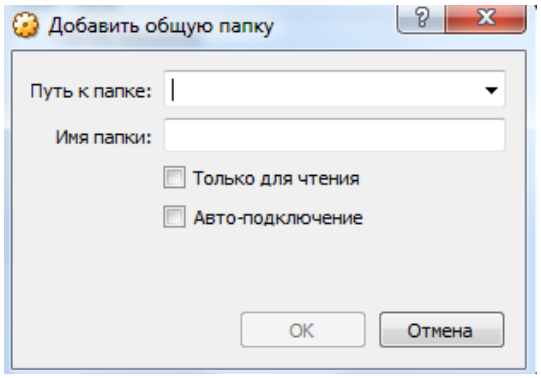


Рисунок А.18 – Окно добавления общей папки

Затем необходимо выбрать папку, которая была предварительно создана в базовой машине. Для этого с помощью стрелки в поле «Путь к папке» выбираем из списка пункт «**Другой**..» и находим нужную нам папку в открывшемся окне. Результат показан на рис. А.19.

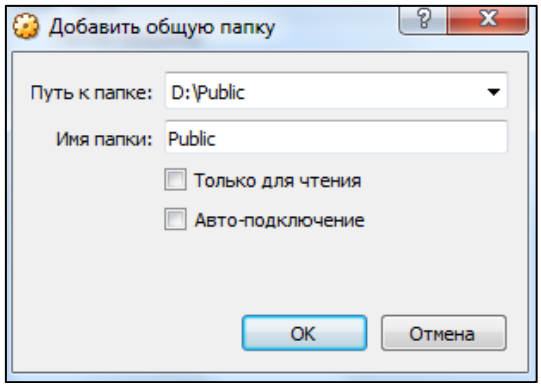


Рисунок А.19 – Выбор общей папки в базовой машине

Поставить маркер в пункт «Авто-подключение».

После нажатия на кнопку «ОК» в окне свойств виртуальной машины должна появится данная папка, как показано на рис. А.20. Нажатие кнопки «ОК» подтверждает окончание действий. Окно свойств после создания общей папки показано на рис. А.20.

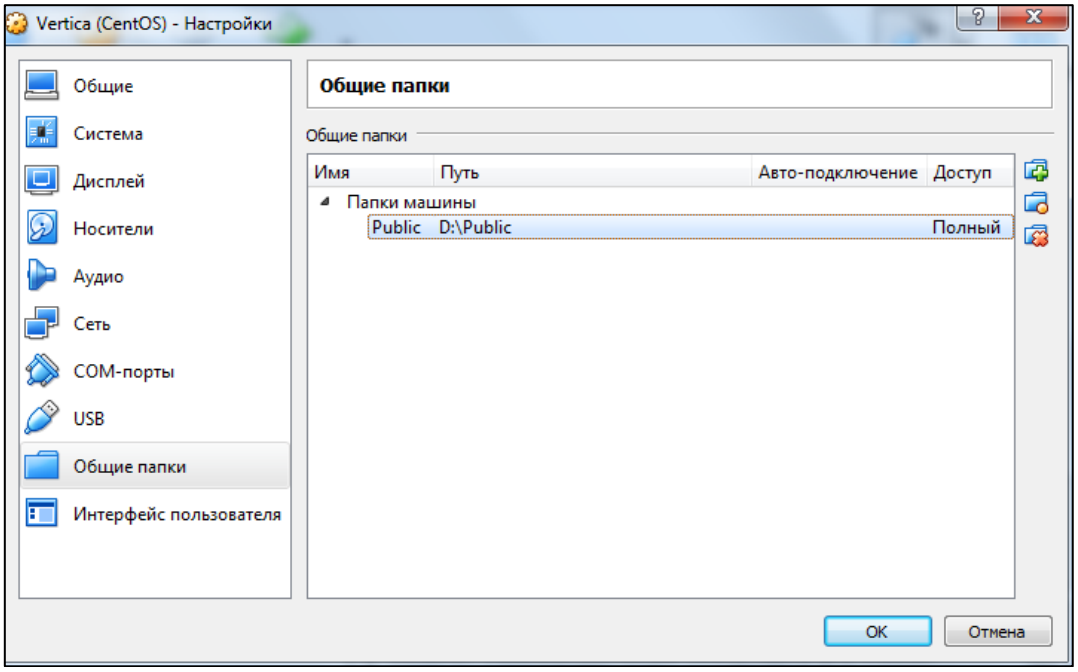


Рисунок А.20 – Окно свойств виртуальной машины после добавления общей папки

Далее надо проверить, что папка носит постоянный характер. Для этого при включенной виртуальной машине нажать на кнопку с желтым кружком «Изменить». Откроется окно, показанное на рис. А.21.

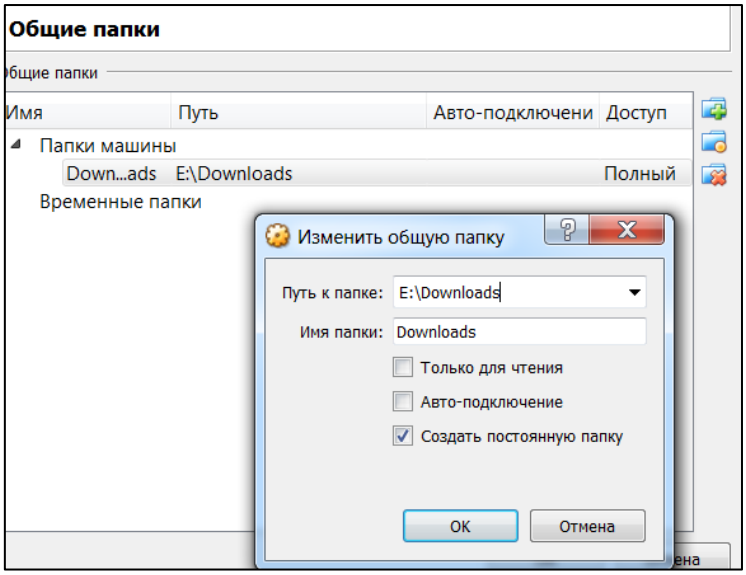


Рисунок А.21 – Свойства общей папки

Поставить галочку в поле маркера «Создать постоянную папку».

Папка в файловом менеджере виртуальной машины отражается в сети и имеет системное имя VBOXSVR, как показано на рис. А.22.

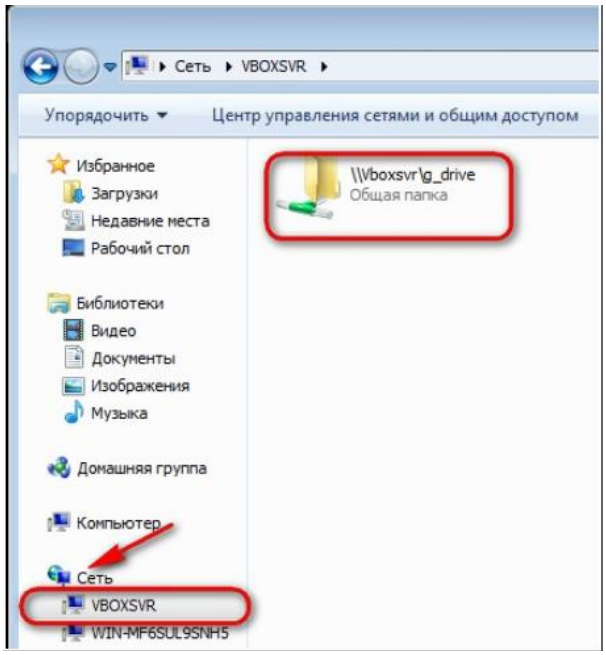


Рисунок А.22 – Общая папка в сети

А.7 Создание образа папки в виртуальной машине в ОС Linux

Настройка общей папки VirtualBox в Linux отличается. Первоначально надо выполнить все предыдущие пункты в основной машине. Включить виртуальную машину и войти в систему. Далее рассмотрены этапы подключения общей папки для пользователя с именем uidbadmin.

Предварительно для удобства работы следует включить общий буфер обмена с основной машиной. Для этого открыть пункт меню “**Устройства**” для приложения VirtualBox, как показано на рис. А.23.

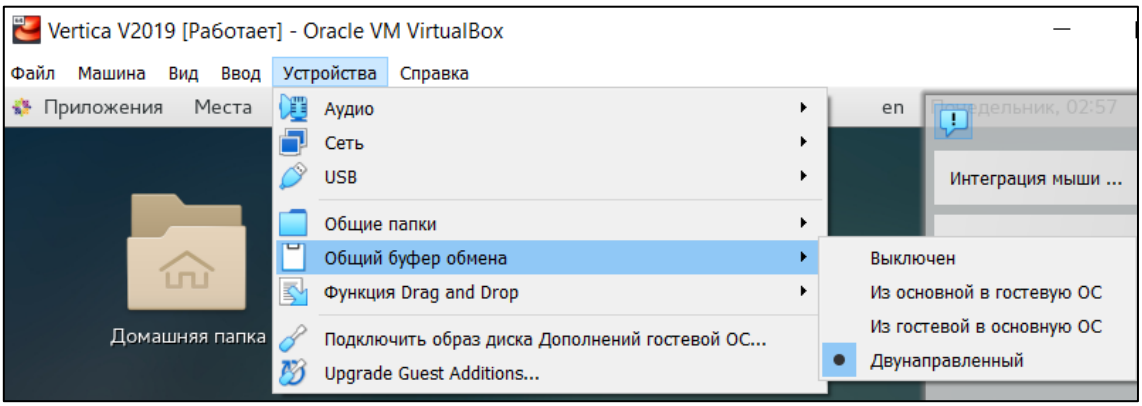


Рисунок А.23 – Включение общего буфера обмена

Зайти в пункт меню «**Общие папки**» и нажать кнопку «**Настроить общие папки.**..», как показано на рис. А.24.

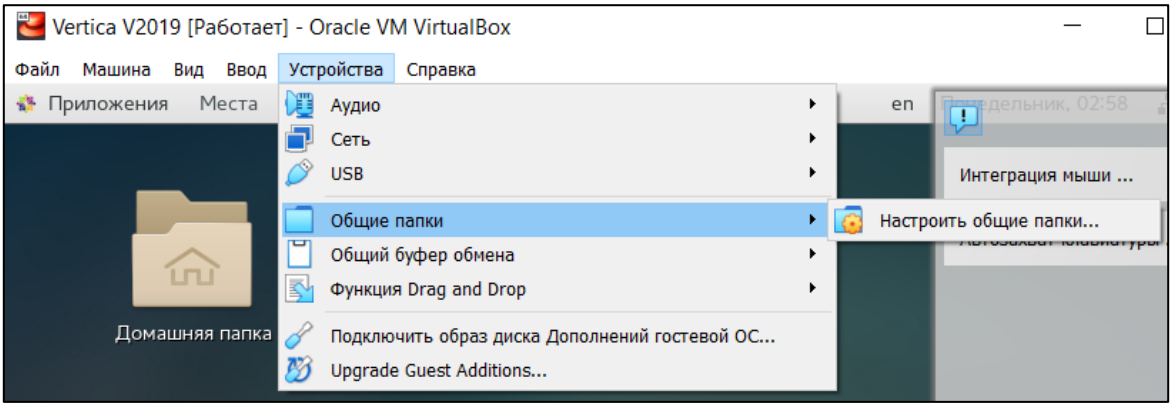


Рисунок А.24 – Переход в режим настройки общих папок

В результате откроется окно со списком общих папок, как показано на рис. А.25. В этом примере имя созданной в основной машине общей папки Vertica-rab.

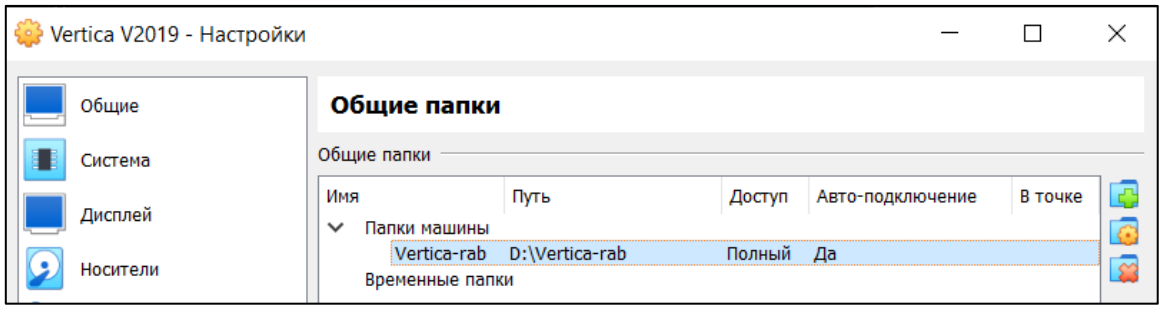


Рисунок А.25 – Список общих папок

Нажать Enter. В результате в виртуальной машине появится папка образа папки в основной машине с именем sf_Vertica-rab, как показано на рис. А.26.

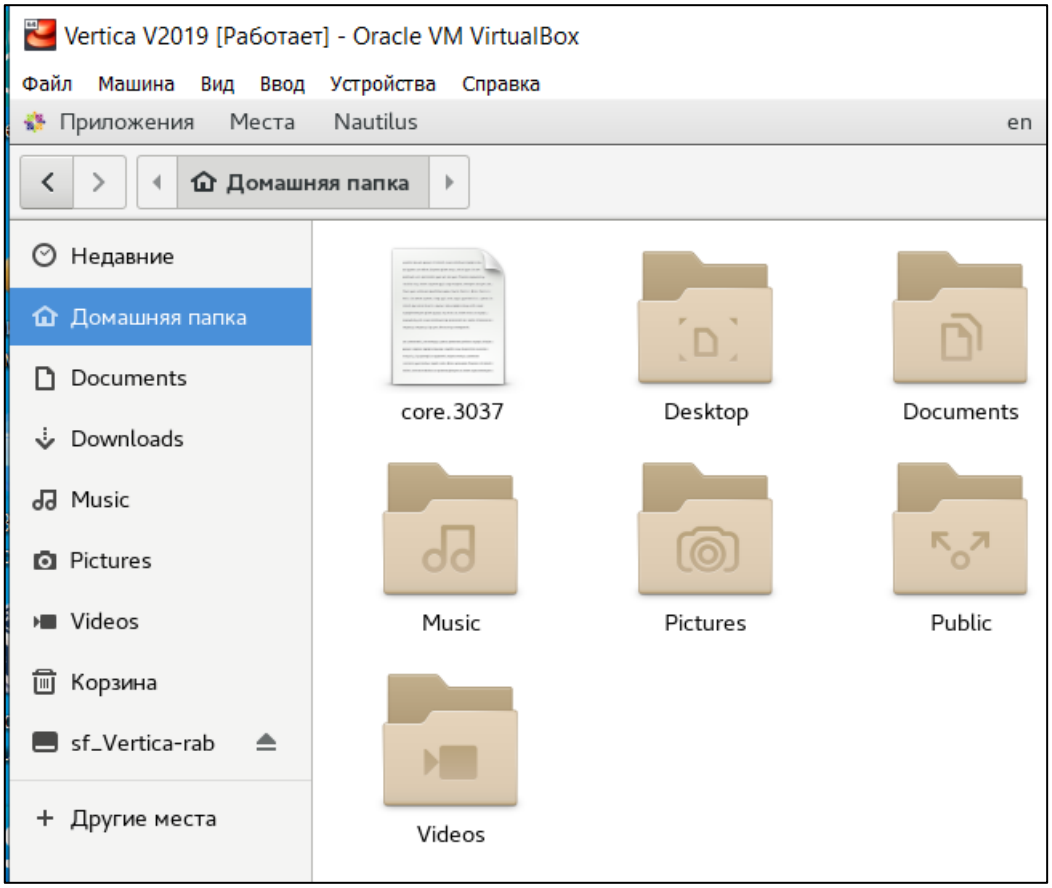


Рисунок А.26 – Новая папка в списке используемых папок

Папка создана, но к ней нет доступа пользователю uidbadmin. При попытке войти в эту папку система выдает сообщение, показанное на рис. А.27.

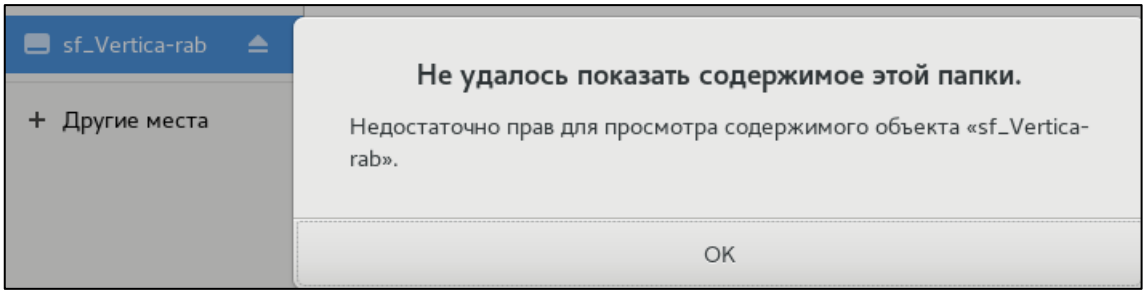


Рисунок А.27 – Сообщение об отказе доступа к папке

Чтобы добавить доступ к образу папки необходимо выполнить ряд команд системы Linux. Первоначально надо зайти в пункт главного меню «**Приложения**». Окно с доступными приложениями показано на рис. А.28.

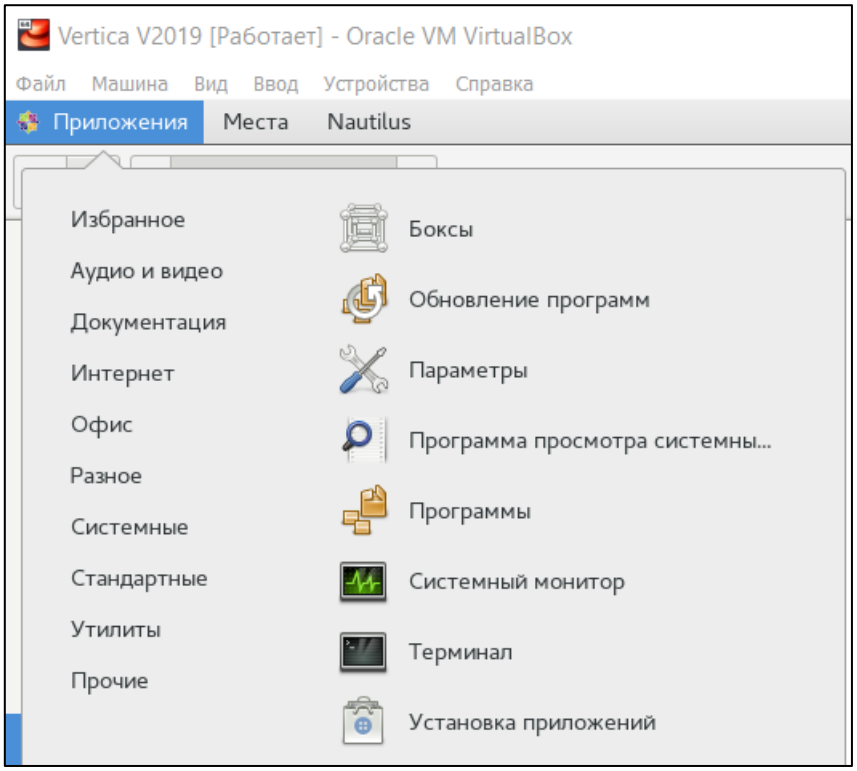


Рисунок А.28 – Список приложений системы

Далее надо открыть пункт «**Терминал»,** при этом откроется окно для ввода команд операционной системы Linux, как показано на рис. А.29.

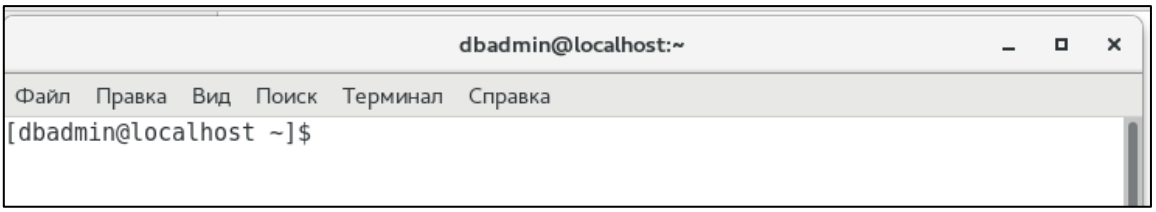


Рисунок А.29 – Окно команд ОС Linux

Первоначально надо выполнить команду для получения пользователю uidbadmin прав администратора. Главным администратором в системе является пользователь с именем root. Для смены пользователя в окне команд надо набрать команду, показанную на рис. А.30.

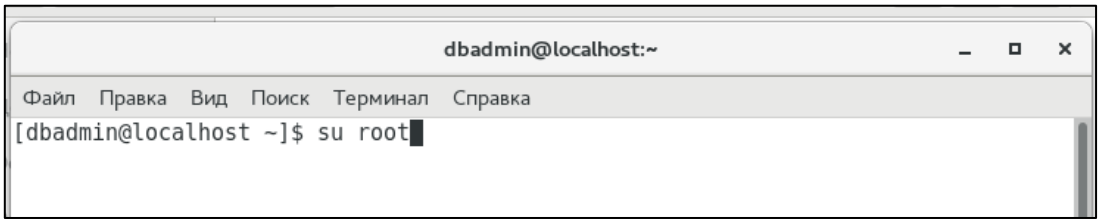


Рисунок А.30 – Команда передачи управления администратору

Система запросит пароль администратора. Пароль в используемой в практикуме системе – password. При этом в Linux при вводе пароля ничего не отображается. Выполнение команды показано на рис. А.31.

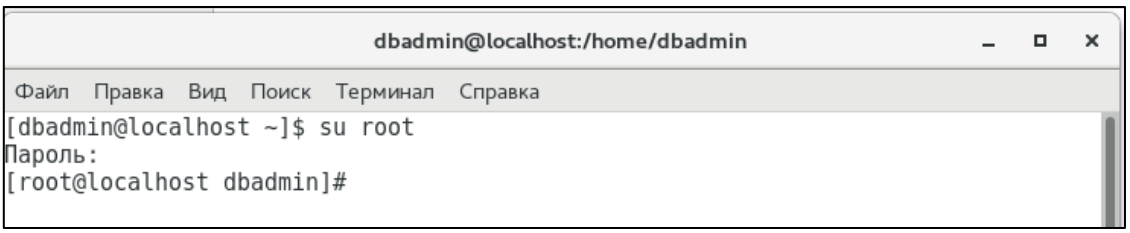


Рисунок А.31 – Администратор готов выполнять команды

Теперь необходимо добавить пользователя в специальный файл, который содержит права пользователей с именем sudoers. Для добавления специальной записи в этот файл можно воспользоваться встроенным текстовым примитивным редактором nano. Команда для вызова этого редактора из командной строки показана на рис. А.32.

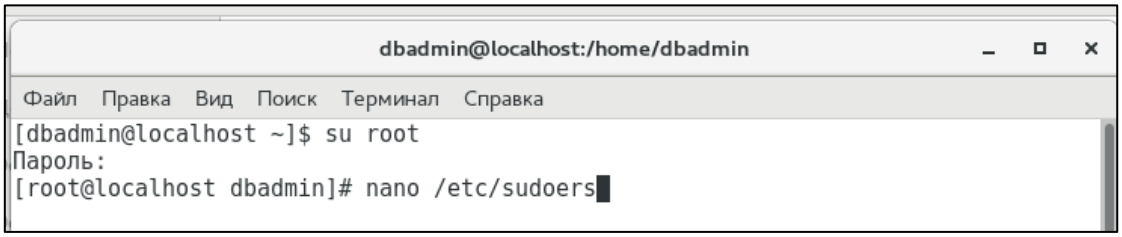


Рисунок А.32 – Вызов текстового редактора из командной строки

В результате откроется окно, показанное на рис. А.33.

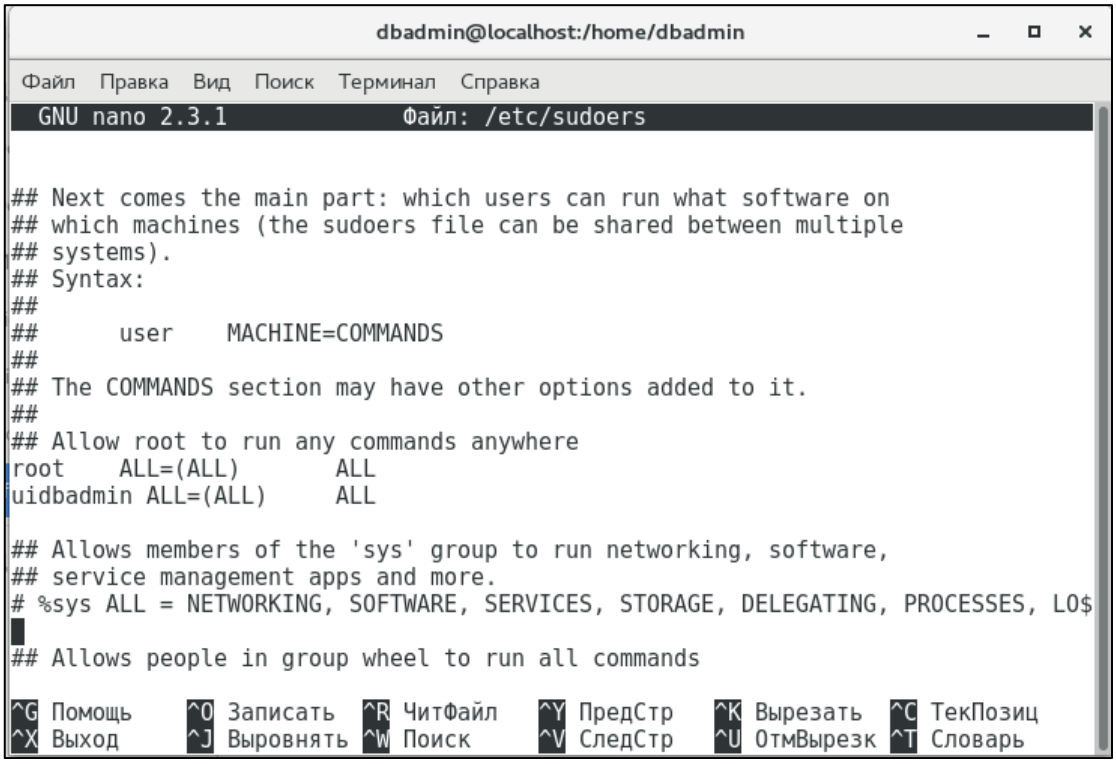


Рисунок А.33 – Окно встроенного текстового редактора

В тексте надо найти строчки для правил доступа и вставить новую строку, как показано на рис. А.33. Значок ^ обозначает клавишу Ctrl на клавиатуре. Выполнить команду Ctrl + O, для записи отредактированного файла и выйти из редактора по набору клавиш Ctrl + X.

Далее нужно добавить нового пользователя в группу vboxsf. Для этого надо выполните следующую команду в Linux:

sudo usermod -aG vboxsf uidbadmin

Вместо uidbadminy, если настройки делаются для пользователя dbadmin (Vertica ADM) нужно указать имя пользователя dbadmin.

После выполнения этой команды нужно перезапустить виртуальную машину. Только после этого ваша общая папка будет работать правильно. На рис. А.34 показана общая папка на рабочем столе в Linux.

Рисунок А.34 – Новая общая папка на рабочем столе

Теперь эту папку можно открыть и пример ее содержимого показан на рис. А.35.

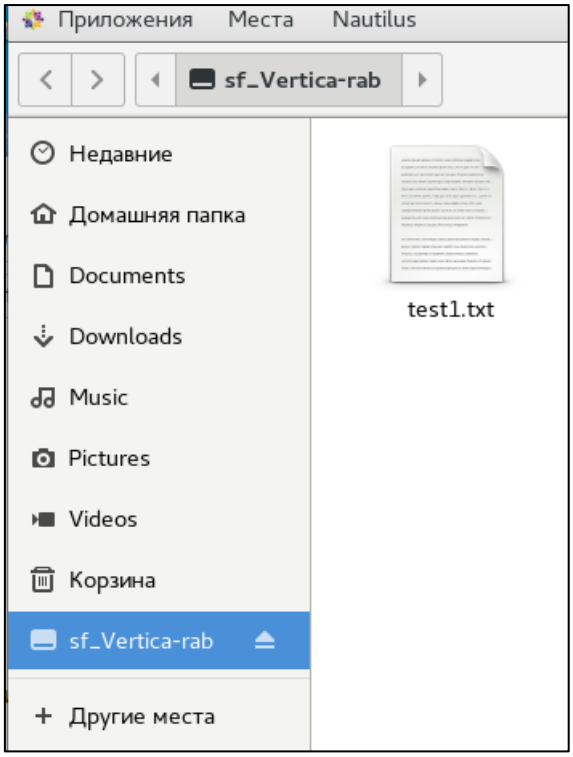


Рисунок А.35 – Созданная общая папка в Linux

А.8 Использование флэш-накопителя в виртуальной машине

Первоначально необходимо загрузить дополнения к системе виртуальных машин. Для этого надо скачать файл с дополнениями с официального сайта (virtualbox.org). На рис. А.36 показан пункт сайта, с которого надо скачивать файл с дополнениями.

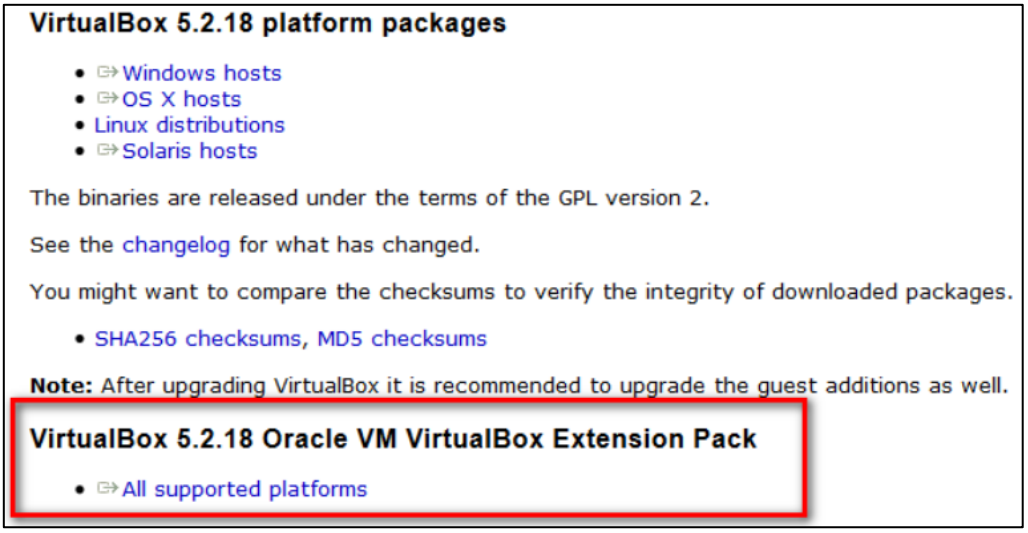


Рисунок А.36 – Пункт на сайте производителя с файлом дополнений

Скачивание надо провести в папку на основной машине. Затем необходимо импортировать дополнения в менеджер виртуальных машин. Для этого надо войти в пункт меню «Файл» окна OracleVM и выбрать пункт «**Настройки**», как показано на рис. А.37.

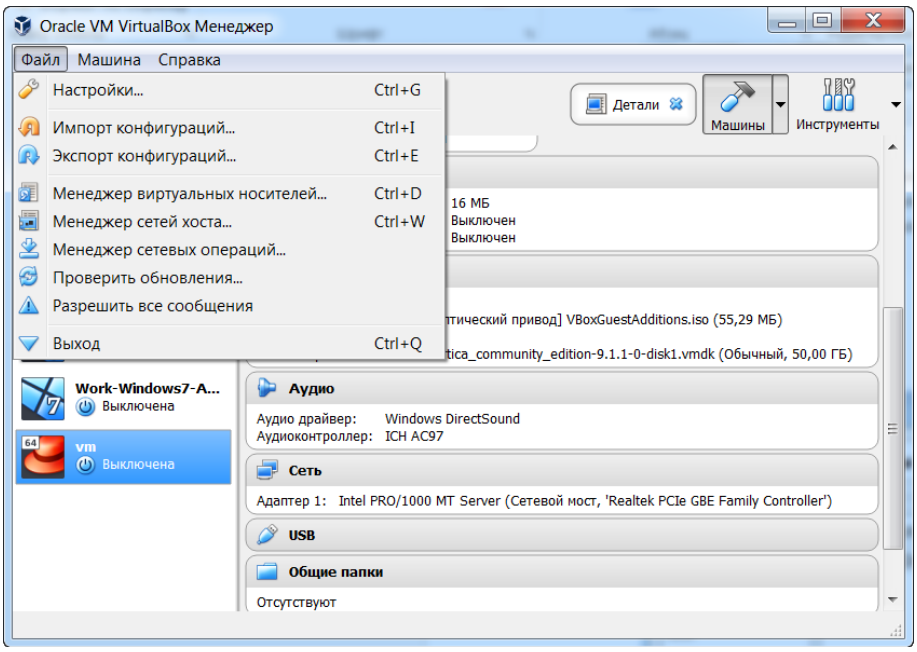


Рисунок А.37 – Выбор пункта меню «Настройки»

В открывшемся окне настроек выбрать пункт меню «Плагины», как показано на рис. А.38.

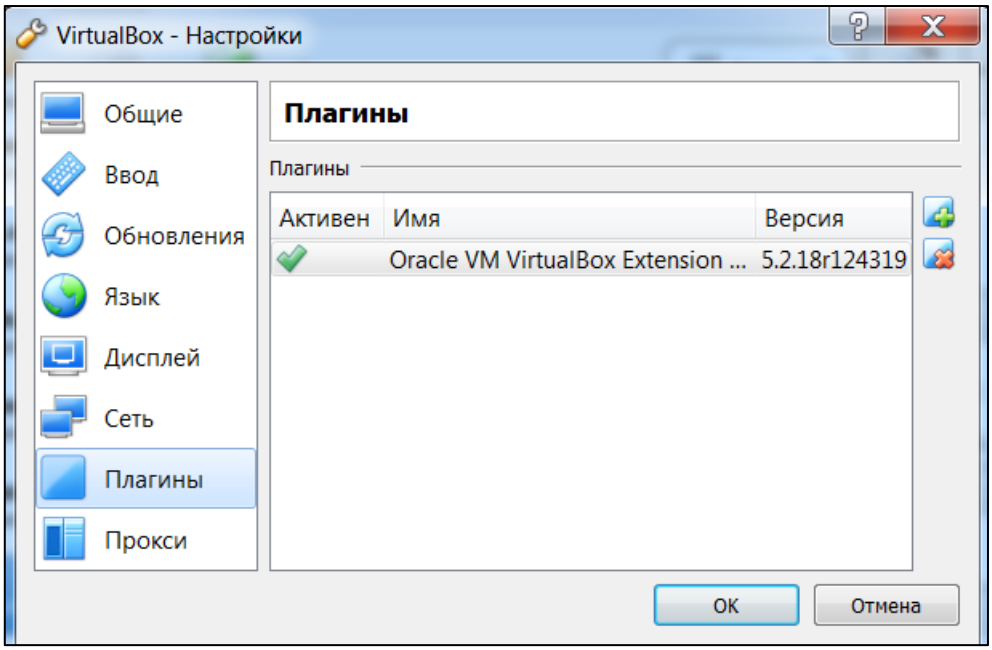


Рисунок А.38 – Пункт меню «Плагины»

Для загрузки дополнений надо нажать кнопку с зеленым крестиком. В результате открывается доступ к файлам основного компьютера. Требуется найти соответствующий файл, как показано на рис. А.27.

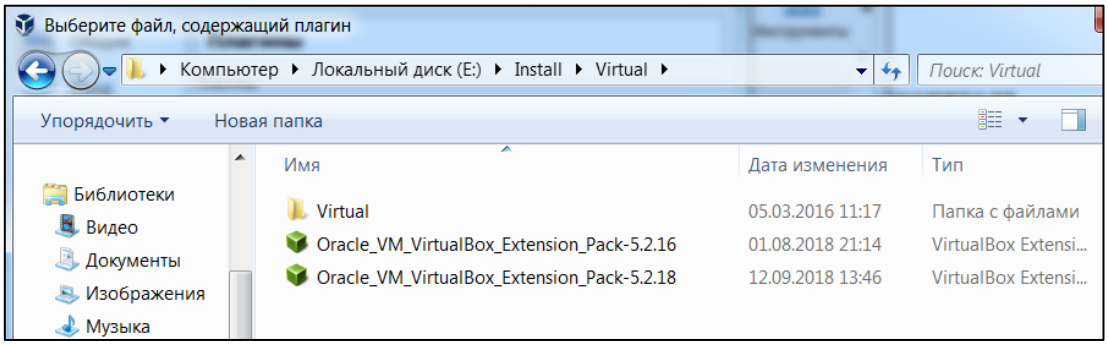


Рисунок А.39 – Поиск файла с дополнениями

Затем надо открыть меню настроек конкретной виртуальной машины, как показано на рис. А.40.

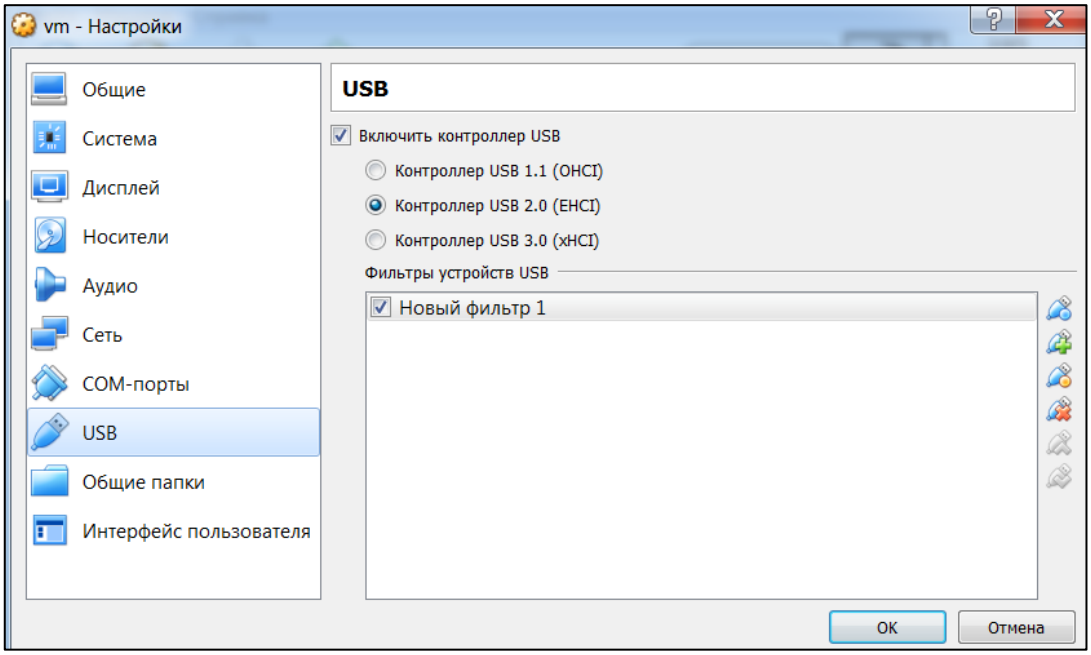


Рисунок А.40 – Настройка драйверов флэшки

Далее надо выбрать соответствующий контроллер флэшки и загрузить фильтр. Лучше всего неопределенный фильтр с помощью кнопки с изображением флэшки с голубым кружком.

Требуется учитывать, что используемая виртуальная машина с системой Linux работает только с флэшками, отформатированными в системе FAT.

Далее необходимо вставить флэшку в разъем компьютера, и перекинуть на нее всю необходимую информацию. Затем нужно запустить виртуальную машину, нажать на пункт «Устройства» и навести курсором на «USB». В выпадающем списке должна быть нужная нам флэшка (рис. А.41).

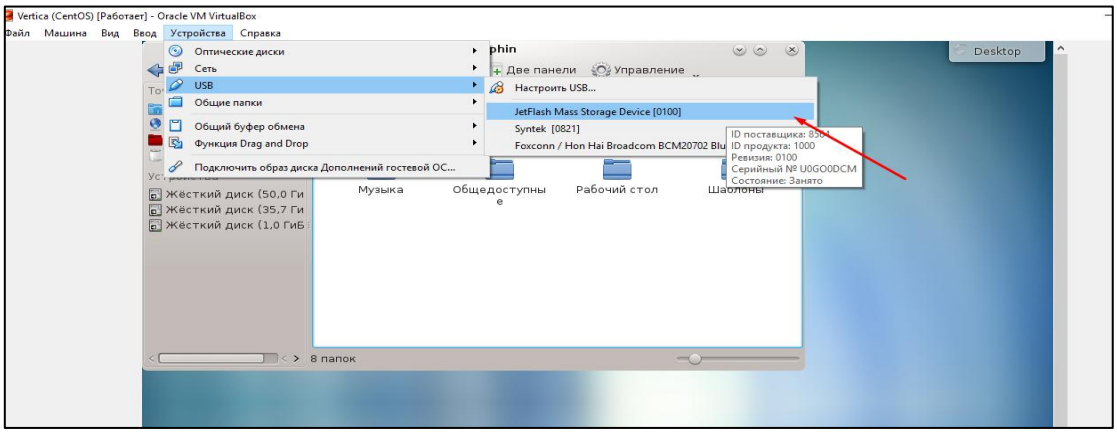


Рисунок А.41 – Выбор флэш-накопителя

После выполнения описанных выше действий флэшка будет подключена к виртуальной машине и появится возможность извлечь с нее необходимые данные (рис. А.42).

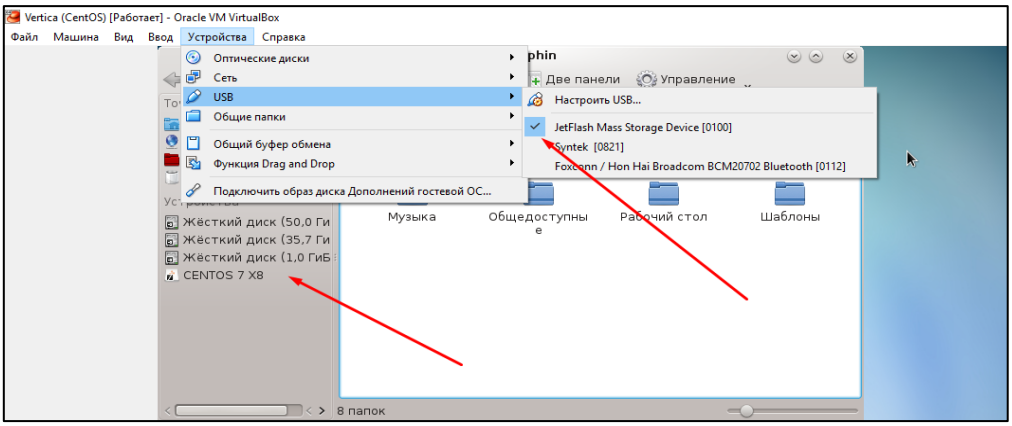


Рисунок А.42 – Подключенный флэш-накопитель

Примечание: при подключении флэшки к виртуальной машине, она становится недоступной для работы с основной ОС. Для того, чтобы вернуть ее для работы с реальной операционной системой, достаточно вынуть ее и вновь вставить, либо же отключить в меню аналогично тому, как выполнялось подключение. Далее можно работать с информацией на флэшке в виртуальной машине. Для активации флэшки в виртуальной машине надо выбрать пункт в активном меню, как показано на рис. А.43.

Рисунок А.43 – Выбор флэш-накопителя

Затем из списка накопителей выбрать нужный накопитель.

Приложение Б

Демонстрационная база данных Vertica

Демонстрационная база данных содержит данные магазина онлайн торговли. База данных VMart содержит следующие три схемы: public, store и online_sales.

Термин «Схема» в Vertica имеет три возможных значения. В операторах SQL термин означает поименованную логическую схему. Логическая схема означает множество таблиц и ограничений. Физическая схема означает множество проекций.

Схема PUBLIC.

На рис. Б.1 показаны все отношения схемы PUBLIC демонстрационной базы данных VMart и связей между ними.

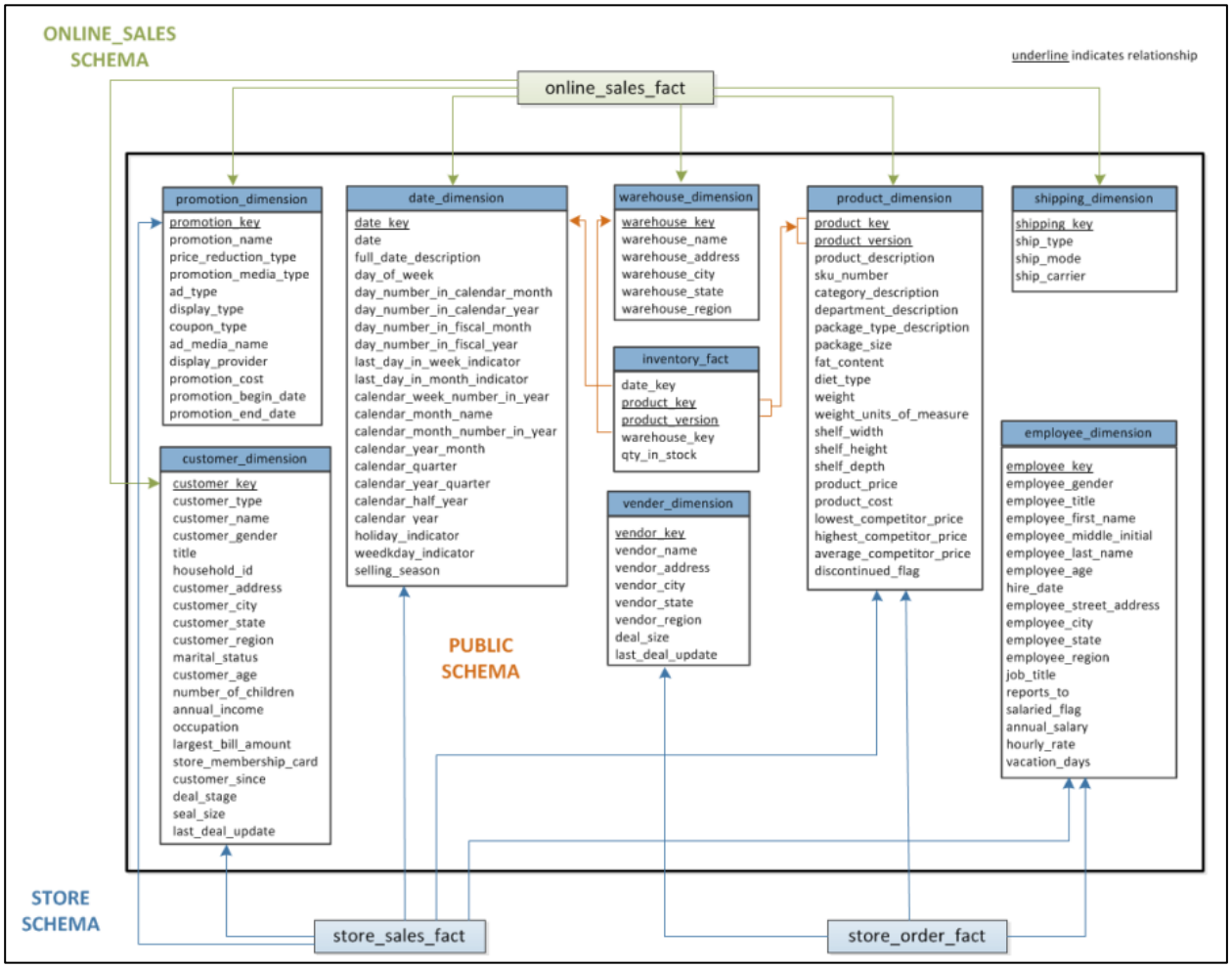


Рисунок Б.1 – Схема PUBLIC демонстрационной базы данных

В табл. Б.1 показан список всех отношений демонстрационной базы данных.

Таблица Б.1. Список всех отношений

public Schema	store Schema	online_sales Schema
inventory_fact	store_orders_ fact	online_sales_fact
customer_ dimension	store_sales_ fact	call_center_ dimension
date_dimension	store_ dimension	online_page_ dimension
employee_ dimension		
product_ dimension		
promotion_ dimension		
shipping_ dimension		
vendor_ dimension		
warehouse_ dimension		

Описание отношений схемы PUBLIC

Отношение **inventory_fact** содержит информацию о количестве продуктов компании на складе.

Column Name	Data Type	NULLs
date_key	INTEGER	No
product_key	INTEGER	No
product_version	INTEGER	No
warehouse_key	INTEGER	No
qty_in_stock	INTEGER	No

Отношение **customer_dimension** содержит первичную информацию обо все клиентах компании.

Column Name	Data Type	NULLs
customer_key	INTEGER	No
customer_type	VARCHAR(16)	Yes
customer_name	VARCHAR(256)	Yes
customer_gender	VARCHAR(8)	Yes
title	VARCHAR(8)	Yes
household_id	INTEGER	Yes
customer_address	VARCHAR(256)	Yes
customer_city	VARCHAR(64)	Yes
customer_state	CHAR(2)	Yes
customer_region	VARCHAR(64)	Yes
marital_status	VARCHAR(32)	Yes
customer_age	INTEGER	Yes
number_of_children	INTEGER	Yes
annual_income	INTEGER	Yes
occupation	VARCHAR(64)	Yes
largest_bill_amount	INTEGER	Yes
store_membership_card	INTEGER	Yes
customer_since	DATE	Yes
deal_stage	VARCHAR(32)	Yes
deal_size	INTEGER	Yes
last_deal_update	DATE	Yes

Отношение **date_dimension** содержит информацию о календарях и датах.

Column Name	Data Type	NULLs
date_key	INTEGER	No
date	DATE	Yes
full_date_description	VARCHAR(18)	Yes
day_of_week	VARCHAR(9)	Yes
day_number_in_calendar_month	INTEGER	Yes
day_number_in_calendar_year	INTEGER	Yes
day_number_in_fiscal_month	INTEGER	Yes
day_number_in_fiscal_year	INTEGER	Yes
last_day_in_week_indicator	INTEGER	Yes
last_day_in_month_indicator	INTEGER	Yes
calendar_week_number_in_year	INTEGER	Yes
calendar_month_name	VARCHAR(9)	Yes
calendar_month_number_in_year	INTEGER	Yes
calendar_year_month	CHAR(7)	Yes
calendar_quarter	INTEGER	Yes
calendar_year_quarter	CHAR(7)	Yes
calendar_half_year	INTEGER	Yes
calendar_year	INTEGER	Yes
holiday_indicator	VARCHAR(10)	Yes
weekday_indicator	CHAR(7)	Yes
selling_season	VARCHAR(32)	Yes

Отношение **employee_dimension** содержит информацию о работниках компании.

Column Name	Data Type	NULLs
employee_key	INTEGER	No
employee_gender	VARCHAR(8)	Yes
courtesy_title	VARCHAR(8)	Yes
employee_first_name	VARCHAR(64)	Yes
employee_middle_initial	VARCHAR(8)	Yes
employee_last_name	VARCHAR(64)	Yes
employee_age	INTEGER	Yes
hire_date	DATE	Yes
employee_street_address	VARCHAR(256)	Yes
employee_city	VARCHAR(64)	Yes
employee_state	CHAR(2)	Yes
employee_region	CHAR(32)	Yes
job_title	VARCHAR(64)	Yes
reports_to	INTEGER	Yes
salaried_flag	INTEGER	Yes
annual_salary	INTEGER	Yes
hourly_rate	FLOAT	Yes
vacation_days	INTEGER	Yes

Отношение **product_dimension** содержит информацию о продуктах компании.

Column Name	Data Type	NULLs
product_key	INTEGER	No
product_version	INTEGER	No
product_description	VARCHAR(128)	Yes
sku_number	CHAR(32)	Yes
category_description	CHAR(32)	Yes
department_description	CHAR(32)	Yes
package_type_description	CHAR(32)	Yes
package_size	CHAR(32)	Yes
fat_content	INTEGER	Yes
diet_type	CHAR(32)	Yes
weight	INTEGER	Yes
weight_units_of_measure	CHAR(32)	Yes
shelf_width	INTEGER	Yes
shelf_height	INTEGER	Yes
shelf_depth	INTEGER	Yes
product_price	INTEGER	Yes
product_cost	INTEGER	Yes
lowest_competitor_price	INTEGER	Yes
highest_competitor_price	INTEGER	Yes
average_competitor_price	INTEGER	Yes
discontinued_flag	INTEGER	Yes

Отношение **promotion_dimension** содержит информацию о рекламе продуктов компании.

Column Name	Data Type	NULLs
promotion_key	INTEGER	No
promotion_name	VARCHAR(128)	Yes
price_reduction_type	VARCHAR(32)	Yes
promotion_media_type	VARCHAR(32)	Yes
ad_type	VARCHAR(32)	Yes
display_type	VARCHAR(32)	Yes
coupon_type	VARCHAR(32)	Yes
ad_media_name	VARCHAR(32)	Yes
display_provider	VARCHAR(128)	Yes
promotion_cost	INTEGER	Yes
promotion_begin_date	DATE	Yes
promotion_end_date	DATE	Yes

Отношение **shipping_dimension** содержит информацию о компаниях доставки продуктов.

Column Name	Data Type	NULLs
shipping_key	INTEGER	No
ship_type	CHAR(30)	Yes
ship_mode	CHAR(10)	Yes
ship_carrier	CHAR(20)	Yes

Отношение **vendor_dimension** содержит информацию о поставщиках продуктов

Column Name	Data Type	NULLs
vendor_key	INTEGER	No
vendor_name	VARCHAR(64)	Yes
vendor_address	VARCHAR(64)	Yes
vendor_city	VARCHAR(64)	Yes
vendor_state	CHAR(2)	Yes
vendor_region	VARCHAR(32)	Yes
deal_size	INTEGER	Yes
last_deal_update	DATE	Yes

Отношение **warehouse_dimension** содержит информацию о пунктах выдачи продуктов и складах.

Column Name	Data Type	NULLs
warehouse_key	INTEGER	No
warehouse_name	VARCHAR(20)	Yes
warehouse_address	VARCHAR(256)	Yes
warehouse_city	VARCHAR(60)	Yes
warehouse_state	CHAR(2)	Yes
warehouse_region	VARCHAR(32)	Yes

Схема STORE.

На рис. Б.2 показаны все отношения схемы STORE демонстрационной базы данных VMart и связей между ними.

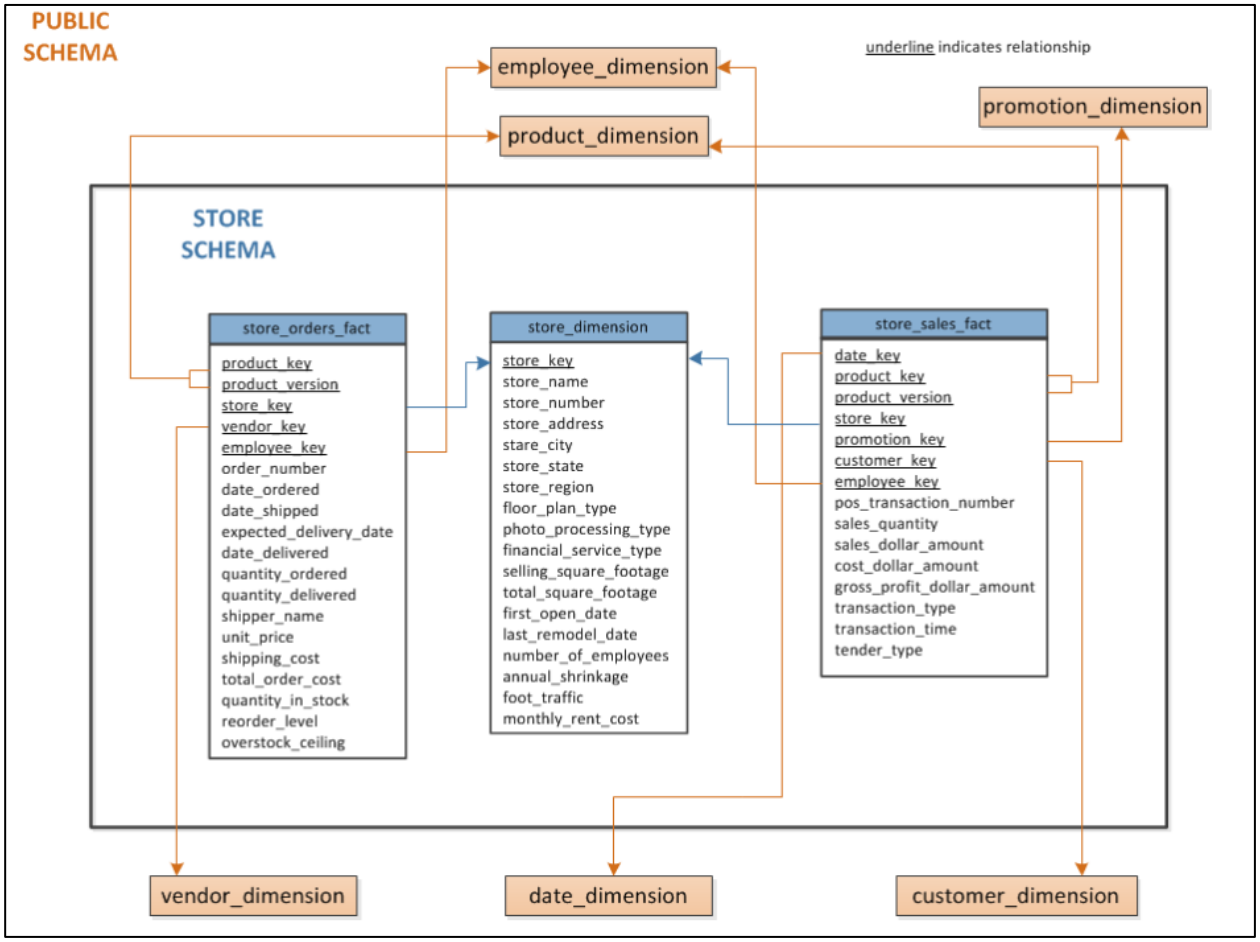


Рисунок Б.2 – Схема STORE демонстрационной базы данных

Эта часть базы данных содержит информацию о региональной сети магазинов и связи этой информации с данными о продуктах, времени и работниках.

Отношение **store_orders_fact** содержит информацию о всех заказах.

Column Name	**Data Type**	**NULLs**
product_key	INTEGER	No
product_version	INTEGER	No
store_key	INTEGER	No
vendor_key	INTEGER	No
employee_key	INTEGER	No
order_number	INTEGER	No
date_ordered	DATE	Yes
date_shipped	DATE	Yes
expected_delivery_date	DATE	Yes
date_delivered	DATE	Yes
quantity_ordered	INTEGER	Yes

quantity_delivered	INTEGER	Yes
shipper_name	VARCHAR(32)	Yes
unit_price	INTEGER	Yes
shipping_cost	INTEGER	Yes
total_order_cost	INTEGER	Yes
quantity_in_stock	INTEGER	Yes
reorder_level	INTEGER	Yes
overstock_ceiling	INTEGER	Yes

Отношение **store_sales_fact** содержит информацию о всех продажах.

Column Name	**Data Type**	**NULLs**
date_key	INTEGER	No
product_key	INTEGER	No
product_version	INTEGER	No
store_key	INTEGER	No
promotion_key	INTEGER	No
customer_key	INTEGER	No
employee_key	INTEGER	No
pos_transaction_number	INTEGER	No
sales_quantity	INTEGER	Yes
sales_dollar_amount	INTEGER	Yes
cost_dollar_amount	INTEGER	Yes
gross_profit_dollar_amount	INTEGER	Yes
transaction_type	VARCHAR(16)	Yes
transaction_time	TIME	Yes
tender_type	VARCHAR(8)	Yes

Отношение **store_dimension** содержит информацию о всех региональных магазинах.

Column Name	**Data Type**	**NULLs**
store_key	INTEGER	No
store_name	VARCHAR(64)	Yes
store_number	INTEGER	Yes
store_address	VARCHAR(256)	Yes
store_city	VARCHAR(64)	Yes
store_state	CHAR(2)	Yes
store_region	VARCHAR(64)	Yes
floor_plan_type	VARCHAR(32)	Yes
photo_processing_type	VARCHAR(32)	Yes
financial_service_type	VARCHAR(32)	Yes
selling_square_footage	INTEGER	Yes
total_square_footage	INTEGER	Yes
first_open_date	DATE	Yes
last_remodel_date	DATE	Yes
number_of_employees	INTEGER	Yes
annual_shrinkage	INTEGER	Yes
foot_traffic	INTEGER	Yes
monthly_rent_cost	INTEGER	Yes

Схема ONLINE_SALES.

На рис. Б.3 показаны все отношения схемы Online_sales демонстрационной базы данных VMart и связей между ними.

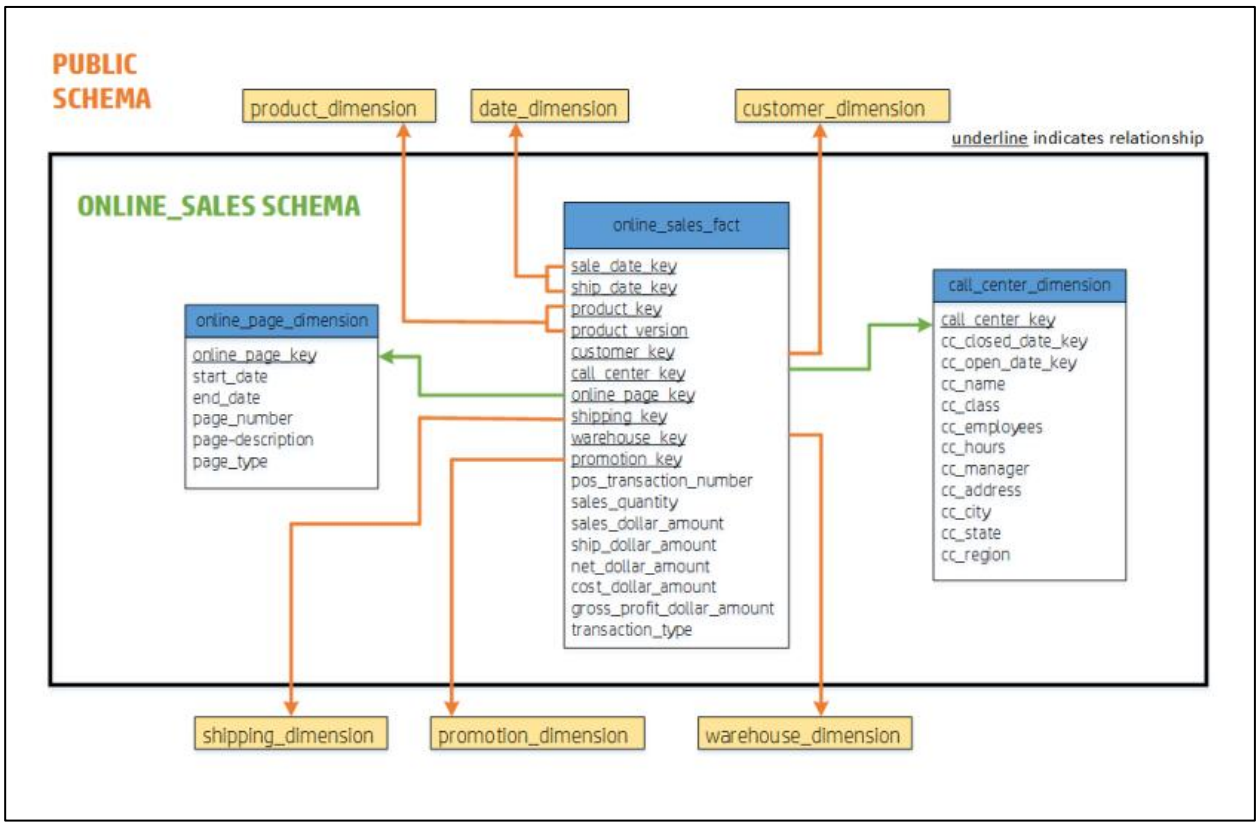


Рисунок Б.3 – Схема ONLINE_SALES демонстрационной базы данных

Схема содержит описание данных о конкретных продажах товаров.

Отношение **online_sales_fact** содержит информацию о всех продажах и связей между товаром, магазином, складом, клиентами и т.д.

Column Name	**Data Type**	**NULLs**
sale_date_key	INTEGER	No
ship_date_key	INTEGER	No
product_key	INTEGER	No
product_version	INTEGER	No
customer_key	INTEGER	No
call_center_key	INTEGER	No
online_page_key	INTEGER	No
shipping_key	INTEGER	No
warehouse_key	INTEGER	No
promotion_key	INTEGER	No
pos_transaction_number	INTEGER	No
sales_quantity	INTEGER	Yes
sales_dollar_amount	FLOAT	Yes
ship_dollar_amount	FLOAT	Yes

net_dollar_amount	FLOAT	Yes
cost_dollar_amount	FLOAT	Yes
gross_profit_dollar_amount	FLOAT	Yes
transaction_type	VARCHAR(16)	Yes

Отношение **call_center_dimension** содержит информацию о всех кол-центрах.

Column Name	Data Type	NULLs
call_center_key	INTEGER	No
cc_closed_date	DATE	Yes
cc_open_date	DATE	Yes
cc_date	VARCHAR(50)	Yes
cc_class	VARCHAR(50)	Yes
cc_employees	INTEGER	Yes
cc_hours	CHAR(20)	Yes
cc_manager	VARCHAR(40)	Yes
cc_address	VARCHAR(256)	Yes
cc_city	VARCHAR(64)	Yes
cc_state	CHAR(2)	Yes
cc_region	VARCHAR(64)	Yes

Отношение **online_page_dimension** должно было содержать информацию о всех сайтах компании, но не доделана разработчиками до конца.

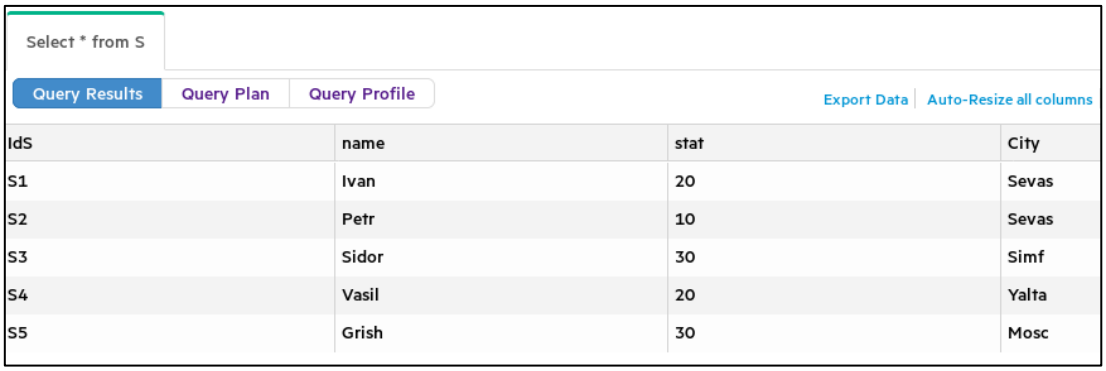
Column Name	Data Type	NULLs
online_page_key	INTEGER	No
start_date	DATE	Yes

База данных Example 1. Схема Public.

Таблица поставщиков. Операторы создания и загрузки данных.

```
CREATE TABLE S (IdS char(5), name char(20), stat int, City char(15));
INSERT INTO     S VALUES  ('S1', 'Ivan', 20, 'Sevas');
INSERT INTO     S VALUES  ('S2', 'Petr', 10, 'Sevas');
INSERT INTO     S VALUES  ('S3', 'Sidor', 30, 'Simf');
INSERT INTO     S VALUES  ('S4', 'Vasil', 20, 'Yalta');
INSERT INTO     S VALUES  ('S5', 'Grish', 30, 'Mosc');
COMMIT;
```

Результат загрузки.



Select * from S

Query Results | Query Plan | Query Profile | Export Data | Auto-Resize all columns

IdS	name	stat	City
S1	Ivan	20	Sevas
S2	Petr	10	Sevas
S3	Sidor	30	Simf
S4	Vasil	20	Yalta
S5	Grish	30	Mosc

Таблица деталей. Операторы создания и загрузки данных.

CREATE TABLE P (IdS char(5), Name char(20), Colour char(7), Weight int, City char(15));

```
INSERT INTO P VALUES ('P1', 'Nut', 'Red', 2, 'Sevas');
INSERT INTO P VALUES ('P2', 'Bolt', 'Green', 7, 'Sevas');
INSERT INTO P VALUES ('P3', 'Screw', 'Blue', 7, 'Simf');
INSERT INTO P VALUES ('P4', 'Screw', 'Red', 4, 'Yalta');
INSERT INTO P VALUES ('P5', 'Cam', 'Blue', 2, 'Mosc');
INSERT INTO P VALUES ('P6', 'Washer', 'Red', 9, 'Simf');
INSERT INTO P VALUES ('P7', 'Star', 'Red', Null, 'Sevas');
COMMIT;
```

Результат загрузки

Query Results | Query Plan | Query Profile

IdP	Name	Colour	Weight	City
P8	Star	Red	4	Sevastopol
P1	Nut	Red	2	Sevas
P2	Bolt	Green	7	Sevas
P3	Screw	Blue	7	Simf
P4	Screw	Red	4	Yalta
P5	Cam	Blue	2	Mosc
P6	Washer	Red	9	Simf
P7	Star	Red		Sevas

Таблица поставок. Операторы создания и загрузки данных.

```
CREATE TABLE SP (IdS char(5), IdP char(5), Quantity int);

INSERT INTO SP VALUES ('S1', 'P1', 300);
INSERT INTO SP VALUES ('S1', 'P2', 200);
INSERT INTO SP VALUES ('S1', 'P3', 400);
INSERT INTO SP VALUES ('S1', 'P4', 200);
INSERT INTO SP VALUES ('S1', 'P5', 100);
INSERT INTO SP VALUES ('S1', 'P6', 100);
INSERT INTO SP VALUES ('S2', 'P1', 300);
INSERT INTO SP VALUES ('S2', 'P2', 400);
INSERT INTO SP VALUES ('S3', 'P2', 200);
INSERT INTO SP VALUES ('S4', 'P2', 200);
INSERT INTO SP VALUES ('S4', 'P4', 300);
INSERT INTO SP VALUES ('S4', 'P5', 400);
COMMIT;
```

Результат загрузки

IdS	IdP	Quantity
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Приложение В

Синтаксис основных предложений языка SQL

Предложение CREATE TABLE

В общем случае предложение имеет следующий формат:

```
CREATE TABLE [IF NOT EXISTS] [[database.]schema.]table
... ( column-definition[,…] [, table-constraint ][,…] )
... [ load-method ]
... [ ORDER BY column[,…] ]
... [ segmentation-spec ]
... [ KSAFE [k-num] ]
... [ partition-clause]
... [ {INCLUDE | EXCLUDE} [SCHEMA] PRIVILEGES ]
```

Элементы предложения в квадратных скобках могут быть опущены.

Значения концептов предложения следующие:

database – Имя базы данных (Может быть опущено);

schema – Имя логической схемы (Может быть опущено);

table – Имя таблицы. Обязательный параметр. Должно быть уникальным в схеме базы данных;

column-definition – Описание столбца таблицы. Таблица может иметь до 1600 столбцов. Описание включает имя столбца, формат данных и ограничения на данные (constraints);

table-constraint – Ограничения на таблицу;

load-method – Метод загрузки данных.

Специальные параметры предложения будут описываться по мере выполнения практикума.

Типы данных SQL в системе Vertica

Тип данных	Размер / bytes	Описание	Сортировка NULL
Двоичные (Binary)			
BINARY	1 to 65,000	Фиксированная длина	NULLS LAST
VARBINARY	1 to 65,000	Переменная длина	NULLS LAST
LONG VARBINARY	1 to 32,000,000	Длинная переменная длина	NULLS LAST
BYTEA/ RAW	Синоним для VARBINARY		
Логические (Boolean)			
BOOLEAN	1	True или False или NULL	NULLS LAST
Длинные символьные (Character Long)			
CHAR	1 to 65,000	Символьная константа фиксированной длины	NULLS LAST
VARCHAR	1 to 65,000	Символьная константа переменной длины	NULLS LAST
LONG VARCHAR	1 to 32,000,000	Длинная символьная константа переменной длины	NULLS LAST
Дата/время (Date/Time)			
DATE	8	Месяц, день и год	NULLS FIRST
TIME	8	Время в течении дня	NULLS FIRST
SMALLDATETIME			
TIME WITH TIMEZONE	8	Время с учетом зоны	NULLS FIRST
DATETIME/TIMESTAMP	8	Дата и время без учета зоны	NULLS FIRST
TIMESTAMP WITH TIMEZONE	8	Дата и время с учетом зоны	NULLS FIRST
INTERVAL	8	Разность между двумя временными точками	NULLS FIRST
INTERVAL DAY TO SECOND	8	Интервал в днях и секундах	NULLS FIRST
INTERVAL YEAR TO MONTH	8	Интервал в годах и месяцах	NULLS FIRST
Действительные числа (Approximate Numeric)			
DOUBLE PRECISION	8	Число с плавающей точкой, 8 байтов	NULLS LAST
FLOAT	8	-:-	NULLS LAST
FLOAT(n)	8	-:-	NULLS LAST
FLOAT8	8	-:-	NULLS LAST
REAL	8	-:-	
Точные цифровые (Exact Numeric)			
INTEGER	8	Целое, 8 байтов со знаком	NULLS FIRST
INT	8	-:-	NULLS FIRST
BIGINT	8	-:-	NULLS FIRST
INT8	8	-:-	NULLS FIRST
SMALLINT	8	-:-	NULLS FIRST

TINYINT	8		NULLS FIRST
DECIMAL	8+	8 байт для первых 18 цифр плюс 8 байт для каждых 19 дополнительных цифр	NULLS FIRST
NUMERIC	8+	-:-	
NUMBER	8+	-:-	NULLS FIRST
MONEY	8+	-:-	NULLS FIRST
Пространственные (Spatial)			
GEOMETRY	1 to 10,000,000	Координаты, как пары (*x*,*y*)	NULLS LAST
GEOGRAPHY	1 to 10,000,000	Координаты, как пары широты и долготы в градусах	
Формат UUID			
UUID	16	16 байт	NULLS FIRST

UUID (universally unique identifier) — это стандартный уникальный идентификатор. UUID представляет собой 16-байтный (128-битный) номер. Общее количество уникальных ключей UUID составляет $2^{128} = 256^{16}$ или около $3{,}4 \times 10^{38}$. Это означает, что, если генерировать 1 триллион ключей каждую наносекунду, то удастся перебрать все возможные значения лишь за 10 миллиардов лет.

Предложение INSERT

В общем случае предложение имеет следующий формат:

```
INSERT [ /*+ hint[, hint] */ ]

 INTO [[database.]schema.]table-name
... [ ( column-list ) ]
... { DEFAULT VALUES | VALUES ( values-list ) | SELECT query-
expression }
```

Элементы предложения в квадратных скобках могут быть опущены.

Значения концептов предложения следующие:

database – Имя базы данных (Может быть опущено);

schema – Имя логической схемы (Может быть опущено);

table-name – Имя таблицы. Обязательный параметр. Должно быть уникальным в схеме базы данных;

column-list – Список атрибутов таблицы.

Специальные параметры предложения в пособии описываются по мере необходимости.

Предложение UPDATE

```
UPDATE [ /*+ hint[, hint] */ ] [[database.]schema.]table-reference
[AS] alias
... SET set-expression [,…]
... [ FROM from-list ]
... [ where-clause ]
```

Дополнительные параметры предложения следующие:

Alias – задает временное имя таблицы.
SET – задает атрибуты, которые должны изменится.
FROM – задает список выражений из других отношений, которые могут использоваться для формирования условий.

Предложение SELECT

В общем случае предложение имеет следующий формат:

```
SELECT [ALL | DISTINCT]
   [<alias>.]<select_item> [AS <column_name>]
   [, [<alias>.]<select_item> [AS <column_name>] ...]
 FROM <relation>[<local_alias>] [,<relation> [<local_alias>] ...]
 [WHERE <joincondition> [AND <joincondition> ...]
  [AND | OR <filtercondition> [AND | OR <filtercondition> ...]]]
 [GROUP BY <groupcolumn> [, <groupcolumn> ...]]
 [HAVING <filtercondition>]
 [UNION [ALL] <SELECT command>]
 [ORDER BY <order_item>[ASC | DESC]
  [, <order_item>[ASC | DESC]...]]
```

Когда используется оператор SELECT для посылки запроса, Vertica интерпретирует запрос и возвращает найденные данные из отношения.

Оператор SELECT запрашивает и специфицирует атрибуты отношения, константы и выражения, появляющиеся в результате запроса.

По умолчанию, все строки, полученные в результате запроса, будут выведены на экран. Для исключения появления одинаковых строк в качестве результата запроса надо включить опцию DISTINCT.

Каждый элемент <select_item> генерирует одну колонку в таблице, формируемой в качестве результата запроса. Если более одного элемента <select_item> имеют одинаковое имя, надо проверить, чтоб для квалификации имени было включено имя псевдонима отношения.

Элемент <select_item> может быть:

- Атрибутом отношения в опции FROM.

- Константой, указывающей что одно и тоже значение появляется в каждой строке результата запроса

- Выражением, которое может включать стандартные функции.

Следующие стандартные функции допустимы для использования с элементом <select_item> (Они являются атрибутами или выражениями включающими атрибуты):

- AVG(<select_item>) – вычисляет среднее арифметическое по столбцу числовых данных.

- COUNT(<select_item>) - вычисляет количество элементов в столбце, соответствующих условию.

- COUNT(*) - счетчик числа строк в выходе запроса.

- MIN(<select_item>) - определяет наименьшее значение элемента <select_item> в столбце.

- MAX(<select_item>) - определяет наибольшее значение элемента <select_item> в столбце.

- SUM(<select_item>) – вычисляет сумму числовых данных в столбце результирующей таблицы.

Необязательная опция AS <column_name> указывает название столбца в таблице на выходе запроса. Это используется, когда элемент <select_item> является выражением или содержит функцию и требуется задать название, имеющее смысл. <column_name> может быть выражением, но не должно содержать символов (например, пробелов) которые недопустимы в именах атрибутов отношения.

Опция FROM <database> [<local_alias>] [, <relation> [<local_alias>] ...].

FROM - обязательная опция, содержит список отношений, в которых находятся данные извлекаемые в результате запроса.

<local_alias> - это временное имя для отношения <relation>. Если указывается псевдоним <local_alias>, необходимо указывать <local_alias> вместо имени отношения везде в команде SELECT. Если несколько баз данных имеют атрибуты с одинаковыми именами необходимо квалифицировать их, для чего может быть использован элемент короткого псевдонима <local_alias>.

Опция [WHERE <joincondition> [AND <joincondition> ...]
[AND | OR <filtercondition> [AND | OR <filtercondition>...]]]

Конструкция запрашивает данные из нескольких отношений. Сообщает системе на необходимость включения в результат запроса только некоторых записей. Условие <joincondition> указывает атрибуты, которые связывают отношения в опции FROM. Если указывается более одного отношения, то необходимо указать условие связи для каждого отношения после первого.

Если включается два отношения в запрос и не указывается условие связи, то каждая запись в первом отношении будет связана с каждой записью во втором отношении пока выполняется условие фильтрации. Это может породить огромный результат запроса.

При нескольких условиях отношения должны объединяться с использованием оператора AND.

Каждое условие <joinconditions> имеет форму:

<field1> <сравнение> <field2>

где <field1> - поле из одного отношения, <field2> - поле из второго отношения и операция <сравнение> принимает одно из следующих значений: =, < >, !=, >, > =, <, < =.

<filtercondition> указывает условие, на основании которого записи включаются в результат запроса. Запрос может включать несколько условий фильтра, связанных операторами AND или OR. Можно использовать также оператор NOT или функцию EMPTY() для проверки пустых полей. Условие фильтра может иметь одну из следующих форм:

Форма: <field1> <comparison> <field2>

Пример: customer.cust_id = payments.cust_id

Форма: <field> <comparison> <expression>

Пример: payments.amount >= 1000

Форма: <field> <comparison> ALL (<subquery>)

Пример: taxrate < ALL ;

(SELECT taxrate FROM customer WHERE state = 'MI')

Когда фильтр включает ключевое слово ALL, <field> должно встретиться в условии сравнения для каждого значения, сгенерированного подзапросом перед включением записи в результат запроса.

Форма: <field> <comparison> ANY | SOME (<subquery>)

Пример: taxrate < ANY ;

(SELECT taxrate FROM customer WHERE state = 'MI')

Когда фильтр включает ключевое слово ANY или SOME, <field> должно встретиться в условии сравнения хотя бы для одного значения, сгенерированного подзапросом перед включением записи в результат запроса.

Форма: <field> [NOT] BETWEEN <start_range> AND <end_range>

Пример: customer.taxrate BETWEEN 5.50 AND 6.00

Проверка, лежит ли значение в диапазоне.

Форма: [NOT] EXISTS (<subquery>)

Пример: EXISTS ;

(SELECT * FROM invoice WHERE customer.zip = invoice.zip)

Когда фильтр включает EXISTS, условие истинно пока запрос не станет пустым.

Форма: <field> [NOT] IN <value_set>
Пример: customer.zip NOT IN ('43411','43506','43667')
<field> -является частью набора значений, перед включением записи в результат запроса.
Форма: <field> [NOT] IN (<subquery>)
Пример: customer.cust_id IN ;
(SELECT tranfile.cust_id FROM tranfile WHERE tranfile.item='Fo’
<field> - является частью набора значений возвращаемых подзапросом, перед включением записи в результат запроса.

Форма: <field> [NOT] LIKE <expC>
Пример: customer.state NOT LIKE 'OH'
Этот фильтр ищет элемент <field>, соответствующий указанному условию <expC>. Можно включить символы процента % и подчеркивания _ как часть условия <expC>. Подчеркивание _ - указывает единичный неопределенный символ в символьной строке, а символ процента % - указывает последовательность неопределённых символов в строке.
Подзапрос - это предложение SELECT внутри другого предложения SELECT и должен быть ограничен скобками. Вы можете организовать множественные подзапросы одного уровня (не вложенные) в опции WHERE. Подзапросы могут иметь множественные условия связи.

[GROUP BY <groupcolumn> [, <groupcolumn> ...]]

Необязательная опция, которая группирует строки в запросе на основании значения в одной или более колонках. Элемент <groupcolumn> может быть регулярным атрибутом отношения, атрибутом отношения которое включено в SQL функцию, или числовым выражением указывающим положение колонки в результирующей таблице. (Номер самой левой колонки).
HAVING <filtercondition>
Сообщает о необходимости включения групп в результат запроса при выполнении условия <filtercondition>. Условие <filtercondition> не может содержать подзапрос.
Элемент HAVING <filtercondition> используется с GROUP BY для указания критерия, по которому группа включается в результат запроса. HAVING может включать так много условий фильтрации, как это необходимо, они связываются операторами AND или OR. Можно также использовать оператор NOT для обращения логических выражений.

HAVING без GROUP BY действует подобно опции WHERE. Вы можете использовать локальные псевдонимы и функции атрибутов в опции HAVING.

[UNION [ALL] <SELECT command>]

Необязательная опция, комбинирует окончательный результат одной команды SELECT с окончательным результатом другой команды SELECT. По умолчанию UNION проверяет комбинацию результатов на предмет исключения повторения строк. Для избежание исключения повторов необходимо указать ключевое слово ALL. Можно использовать скобки для комбинирования нескольких опций UNION.

UNION правила:

■ Нельзя использовать UNION для комбинации подзапросов.

■ Обе команды SELECT должны выводить одинаковое число колонок.

■ Каждая колонка в результате запроса одной команды SELECT должна иметь тот же тип данных, что и соответствующая колонка в другой команде SELECT.

■ Только последняя команда SELECT может иметь опцию ORDER BY и ORDER BY должна описывать выходную колонку по номеру. Если ORDER BY включена, эффект сказывается на всех результатах.

[ORDER BY <order_item>[ASC | DESC]
[, <order_item> [ASC | DESC]...]]

Сортирует результат запроса на основании одной или нескольких колонок. Каждый элемент <order_item> соответствует колонке в результате запроса и может быть:

■ Полем из FROM отношения, которое также является элементом <select_item> в главной опции SELECT (не в подзапросе)

■ Числовым выражением, указывающим положение колонки в результирующей таблице.

Можно указать необязательный ключ DESC, для сортировки результата в убывающем порядке в соответствии с <order_item(s)>. ASC указывает на порядок по возрастанию и принимается по умолчанию. Если опцию не указывать результат будет неупорядоченным.

Учебное издание

Цуканов Александр Викторович
Русина Наталия Анатольевна

Поиск зависимостей (Data Mining) в больших базах данных

Лабораторный практикум

В авторской редакции

Изд. № 67/2025. Объем 9,75 п.л. Усл. печ. л. 9,07. Уч.-изд. л. 9,555.
Издательский центр ФГАОУ ВО «Севастопольский государственный университет»