АДМИНИСТРИРОВАНИЕ И КИБЕРБЕЗОПАСНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ

Екатеринбург
2022

Министерство науки и высшего образования Российской Федерации
Уральский государственный экономический университет

АДМИНИСТРИРОВАНИЕ И КИБЕРБЕЗОПАСНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ

Рекомендовано
Советом по учебно-методическим вопросам и качеству образования
Уральского государственного экономического университета
в качестве учебного пособия

Екатеринбург
2022

УДК 004.9(075.8)
ББК 32.81я73
А28

Рецензенты:

Институт радиоэлектроники и информационных технологий — РТФ
Уральского федерального университета
имени первого Президента России Б. Н. Ельцина
(протокол № 1 от 1 сентября 2022 г.);

генеральный директор ООО «Октоника»
К. Г. Ведьманов

Авторский коллектив:

В. П. Часовских, Г. А. Акчурина, В. Г. Лабунец, Е. Н. Стариков, Е. В. Кох

А28 **Администрирование и кибербезопасность информационных систем :** учебное пособие / В. П. Часовских, Г. А. Акчурина, В. Г. Лабунец [и др.] ; Министерство науки и высшего образования Российской Федерации, Уральский государственный экономический университет. — Екатеринбург : УрГЭУ, 2022. — 173 с.

В учебном пособии определяются общетеоретические аспекты формализации и использования различных моделей (от традиционных до новейших интеллектуальных систем) администрирования и кибербезопасности информационных систем для frontend- и backend-разработки. Определяются пути создания более экономных, оптимизированных под облачные технологии и адаптируемых к функционированию на мобильных устройствах веб-приложений для платформы .NET.

Структура и содержание учебного пособия полностью соответствуют требованиям федеральных государственных образовательных стандартов высшего образования (ФГОС ВО) и учебной программе дисциплины «Администрирование информационных систем» и «Интеллектуальные технологии и кибербезопасность».

Для студентов очной и заочной форм обучения направлений 02.03.03 «Математическое обеспечение и администрирование информационных систем» (бакалавриат), 09.03.03 «Прикладная информатика» (бакалавриат), 09.03.01 «Информатика и вычислительная техника» (бакалавриат), 09.04.03 «Прикладная информатика» (магистратура).

УДК 004.9(075.8)
ББК 32.81я73

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Стратегия развития информационного общества в Российской Федерации на 2017–2030 гг. направлена на создание условий для развития общества знаний в Российской Федерации, повышение благосостояния и качества жизни граждан нашей страны путем повышения доступности и качества товаров и услуг, произведенных в цифровой экономике с использованием современных цифровых технологий, повышения степени информированности и цифровой грамотности, улучшения доступности и качества государственных услуг для граждан, а также безопасности как внутри страны, так и за ее пределами. Важное место в цифровой экономике занимают информационные системы (ИС).

Цель учебного пособия — формирование системы теоретических и практических знаний о современных технологиях администрирования информационных систем и создание средств защиты от кибератак. Администрирование информационной системы рассматривается как предоставление пользователям соответствующих прав использования возможностей работы с системой (базой данных); обеспечения целостности данных, а также создания многопользовательских приложений. Выделяются три основные категории пользователей информационной системы: разработчики, администраторы и собственно пользователи. Администрирование информационной системы осуществляется лицом, управляющим этой системой — администратором.

Информационные системы появились примерно полвека назад, и до недавнего времени основой ИС, как правило, считалась база данных (БД), теория, методы и средства которой были разработаны и внедрены в ИС в середине 60-х гг. XX в.

Для управления БД используется система управления базами данных (СУБД). Традиционно администрирование ИС неотделимо от администрирования СУБД.

Функции администратора СУБД разнообразны. Конечно, первое, что должен уметь такой специалист — это установить сервер

баз данных и выполнить его первоначальную настройку. Другим важным компонентом квалификации администратора является хорошее знание языка структурированных запросов (Structured Query Language — SQL) к данным БД, в том числе умение оптимизировать запросы к базе данных с точки зрения быстроты их выполнения. Администратор должен хорошо представлять себе эти механизмы. Он отвечает за работу сервера БД, но на этом сервере хранятся не только данные пользователей, но и так называемые хранимые процедуры. Эти процедуры разрабатываются на различных языках и используются в различных ситуациях для повышения производительности работы прикладных программ. Администратор должен хорошо понимать принципы их работы.

За последние 10 лет достижения в создании вычислительных машин и их повсеместном применении определило то, что названо четвертой промышленной революцией. В нашей стране была принята Национальная программа «Цифровая экономика РФ», утвержденная протоколом заседания президиума Совета при Президенте Российской Федерации по стратегическому развитию и национальным проектам от 4 июня 2019 г. № 7. В этой программе указано, что ключевым фактором производства являются данные в цифровом виде, обработка больших объемов и использование результатов анализа которых по сравнению с традиционными формами хозяйствования позволяют существенно повысить эффективность различных видов производства, технологий, оборудования, хранения, продажи, доставки товаров и услуг.

Сегодня под цифровой экономикой в широком смысле принято понимать цифровую трансформацию всей существующей экономики, основой которой служат цифровые технологии. В программе определены девять сквозных технологий: большие данные (Big Data); нейротехнологии и искусственный интеллект; системы распределенного реестра; квантовые технологии; новые производственные технологии; промышленный интернет; компоненты робототехники и сенсорика; технологии беспроводной связи; технологии виртуальной и дополненной реальностей. Была принята национальная стратегия развития искусственного интеллекта на период до 2030 г., в которой определено, что развитие ИС, помогающих человеку принимать решения, предполагает замену экспертных систем системам машинного обучения.

Ключевым фактором перечисленных изменений являются данные в цифровом виде и обработка больших объемов данных.

Цифровая экономика, сквозные технологии и прежде всего Big Data определили беспрецедентные преобразования в индустрии баз данных. Жизненные циклы внедрения технологий БД ускорились. Архитектура БД развивается столь быстро, что привычные задачи уже не требуют решения, а связанные с ними навыки утратили актуальность.

Сквозные технологии цифровой экономики, новые требования кибербезопасности, концепция инфраструктуры как кода (Infrastructure as Code, 1aC), возможности облачных технологий определили необходимость изменения разработки и администрирования БД.

Происходит переход от традиционных административных задач к процессу, в котором основное внимание уделяется архитектуре, автоматизации, разработке программного обеспечения (в широком смысле), непрерывной интеграции и распространению, средствам мониторинга и обслуживания систем. При этом ценность и значимость данных увеличились как минимум на порядок, у Big Data появилась стоимость.

Перечисленные изменения определяют новые методы и средства ИС, БД и компетенции администрирования ИС.

Будем рассматривать ИС как совокупность средств и методов, предназначенных для хранения, обработки, поиска и предоставления информации конечному пользователю, как результат обработки Big Data. Специалиста (или группу специалистов), создающего и эксплуатирующего такую ИС организации, будем называть администратором системы с функциями «дата-инжиниринг» (Data Engineering).

В данном учебном пособии рассматриваются основные темы из области администрирования ИС, предполагающие, что оно заключается в предоставлении пользователям соответствующих прав использования возможностей работы с системой (базой данных, нейронной сетью); обеспечении целостности данных, а также создании многопользовательских приложений. Администрирование ИС — это ее установка, управление доступом к ней, обеспечение целостности ИС и др.

Установка ИС на компьютере заключается в подготовке одного или нескольких файлов (библиотек) данных, которые будут установлены и использоваться в виде единой ИС. Рассматривается новая концепция для управления и описания инфраструктуры центра обработки данных через конфигурационные файлы (Infrastructure as Code, IaC) и изменение роли администратора ИС. Была показана необходимость замены традиционных административных задач баз данных процессом, в котором основное внимание уделяется архитектуре, автоматизации, разработке программного обеспечения (в широком смысле), непрерывной интеграции и распространению, средствам мониторинга и обслуживания систем.

Кибербезопасность (ее иногда называют компьютерной безопасностью) рассматривается как совокупность методов и практик защиты от атак злоумышленников для компьютеров, серверов, мобильных устройств, электронных систем, сетей и данных. В учебном пособии выделяются несколько основных категорий. Безопасность приложений — защита устройств от угроз, которые преступники могут спрятать в программах. Зараженное приложение может открыть злоумышленнику доступ к данным, которые оно должно защищать. Безопасность информации — обеспечение целостности и приватности данных как во время хранения, так и при передаче. Операционная безопасность — к этой категории относятся управление разрешениями для доступа к сети или правилами, которые определяют, где и каким образом данные могут храниться и передаваться.

В учебном пособии администрирование ИС будет демонстрироваться с использованием MS SQL-2019 и SQL СУБД Greenplum, популярной для Big Data.

Для данных на сервере будем использовать системы управления базами данных Microsoft SQL и операционную систему Microsoft Server — 2019.

Основная задача учебного пособия — дать студентам общие систематизированные сведения об организации и структуре ИС. В доступной форме даны базовые понятия, цели, задачи администрирования ИС, принципы и правила настройки и использования стека протоколов TCP/IP, методы настройки различных типов адресации в IP-сетях. Предполагается рассмотрение эф-

фективных решений задач управления пользователями и ресурсами сети, а также приобретение необходимых знаний и навыков в области кибербезопасности функционирования ИС.

В учебном пособии рассмотрены общие принципы и подходы использования в ASP.NET Core MVC 6 платформы Ruby on Rails, предложившей революционную технологию применения архитектуры MVC совместно с протоколом HTTP. Рассматривается использование JavaScript в качестве основного языка программирования.

Особенностью данного учебного пособия является системное последовательное представление разработки и администрирования ИС с учетом проблем разработки ИС, моделей данных веб-технологий в разрезе их эволюции. Новизной обладает сравнительный анализ администрирования ИС и выбранной среды обеспечения кибербезопасности. Существенную часть занимают вопросы практической реализации администрирования и кибербезопасности в среде современных веб-технологий.

Структура учебного пособия включает основной теоретический материал и обобщенные выводы по нему в конце каждой главы. С целью повышения эффективности освоения материала по каждой главе сформулированы вопросы для самоконтроля. Авторы учебного пособия рекомендуют обучающимся после изучения теории для закрепления пройденного материала формулировать ответы на предложенные вопросы. Приведены небольшие практические задания, которые рекомендуется выполнить на основе примеров, разобранных в основном материале главы.

ГЛАВА 1

BIG DATA информационные системы: данные, технологии, аналитика, люди

Данные, технологии, аналитика и люди — это комплекс, можно сказать, экономическое явление, а не технология.

Большие данные отличаются от просто данных не только своим объемом. Они также отличаются и тем, что для работы с ними нужны специальные инструменты. То есть просто так сесть за компьютер, поработать с большими данными не удастся, необходима специальная инфраструктура.

1.1. Источники больших данных

Изучая особенности больших данных, важно понимать, что большие данные — это данные оцифрованные, т. е. все, что находится в папках и в «цифру» не переведено — это небольшие данные. Источников больших данных достаточно много. Это может быть информация о счетах и переводах; это могут быть истории ваших перемещений в сети Интернет; это могут быть данные с датчиков; это может быть ГЛОНАСС-сигнал, данные с видеокамер в логистических центрах, на автомобильных дорогах, в аудиториях университетов и т. д.; это можем быть мы с вами — физические лица, которые что-то совершают и производят, и это, безусловно, государство. Государство является крупнейшим держателем данных, и, соответственно, если говорить о способах получения данных для обработки, то их несколько. Во-первых, это может быть собственная генерация, когда компания обрабатывает какие-то операции, формирует информацию. Информация может собираться из открытых источников: есть предложение скачать прогноз погоды на месяц, покупка данных. Есть компании, которые продают датасет, и может быть история, связанная с партнерством, когда совместно с кем-то идет об-

работка данных, предоставляется продукт, формируются новые данные. В результате такой кооперации компания становится совладельцем таких данных. Если говорить о том, что данные — это уже объект управления, то это означает, что необходимо регулировать работу с ними. Актуальной является работа с персональными данными: как предотвращать утечки, какими должны быть юридические последствия за неправомерное разглашение третьим лицам персональных данных и т. д. Эти вопросы сейчас активно обсуждаются и уже входят в сферу интересов специалистов, которые работают с большими данными. На рис. 1 показаны различные данные.

Рис. 1. Данные различной структуры

1.2. Инфраструктура для формирования больших данных

Для формирования данных необходима базовая инфраструктура (рис. 2) — это оборудование для сбора данных, сеть для передачи данных, сервер, который обрабатывает эту инфор-

мацию и производит запись. Базовая инфраструктура — это оборудование, которое должен представлять администратор ИС.

Рис. 2. Формирование Big Data

Следующая составляющая — это цифровая инфраструктура, включающая платформы и инструменты, которые позволяют разместить на хранение данные сервера, возможно их разблокировать, чтобы корректно хранить и обрабатывать. И это все — платформа для работы с большими данными.

Если мы говорим о данных, есть две базовые функции: данные можно хранить и можно обрабатывать. По итогам обработки можно получить результат, и этот результат формируется технологическими инструментами. Есть инструменты специальной работы с большими данными, с помощью которых производится базовая аналитика.

И завершающий уровень — вертикальные решения и услуги. Это прикладное решение из данных, которое позволяет пользователю получить какую-то новую для него информацию, например, курс доллара.

Базовая инфраструктура (оборудование и сети) обычно в сферу рынка больших данных не включается. Потому что они могут использоваться и для других нужд, например, для разговора по телефону, для серфинга сети Интернет и так далее. На рис. 3 показан фрагмент базовой инфраструктуры.

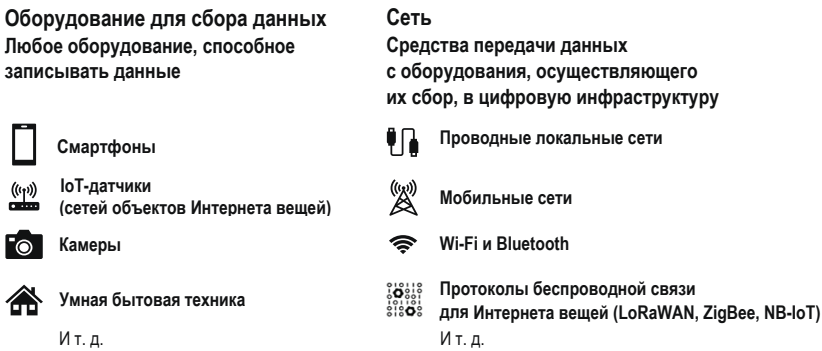

Рис. 3. Фрагмент базовой инфраструктуры

А вот уже цифровая инфраструктура (рис. 4) обеспечивает прием хранения обработку данных. То есть это серверы, это облачные среды, программное обеспечение, которое позволяет хранить и обрабатывать данные. Вот это уже предмет интереса рынка больших данных. То есть все, что касается инструментов и средств, можно сказать, производства для работы с данными — это уже рынок больших данных.

Рис. 4. Цифровая инфраструктура

Если говорить о прикладных инструментах (рис. 5), базовые аналитики — это, безусловно, основа рынка больших данных, поскольку они развиваются достаточно интенсивно и позволяют решать все более и более сложные задачи. Поэтому мы на технологические инструменты также смотрим как на приложения для решения конкретных задач.

Рис. 5. Прикладные инструменты

Разработка вертикальных приложений (рис. 6) является перспективным направлением, потому что задач становится все больше, и можно предлагать различные продукты из SIERVICE. И это огромный потенциал для развития рынка больших данных. Можно сказать, на разных уровнях — и на локальных, и на глобальных — выделяется четыре управленческих аспекта. Мы смотрим на данные, мы смотрим на технологии.

Если мы рассматриваем вопросы управления данными (рис. 7), то данные должны быть сформированы таким образом, чтобы на их основе можно было принимать решение. Иначе смысла хранить данные практически нету. Мы должны смотреть на данные как на возможный источник дохода — есть компании, которые собирают данные, формируют из них продукты, продают и на этом зарабатывают. Необходимо смотреть на информацию как на актив, который нужно защищать, приращивать и делать его можно более ценным.

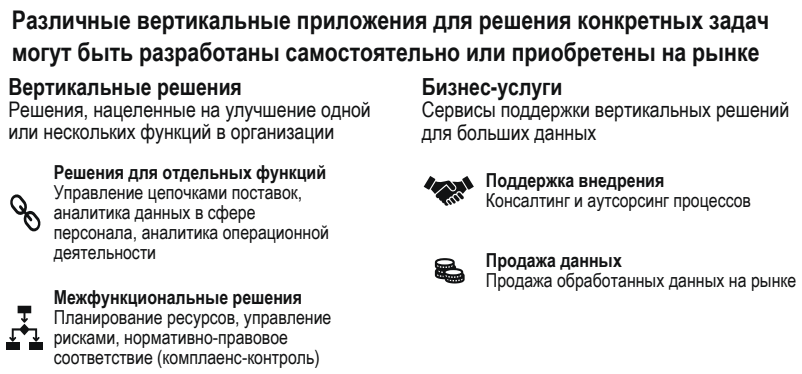

Рис. 6. Вертикальные приложения

Рис. 7. Большие данные как объект управления

В процессе управления ассортиментом данных необходимо понимать, какие возникают вопросы, на которые необходимо получить ответы, и для этого подобрать все необходимые данные, чтобы ответ был максимально полным. Надо также понимать, что решение нужно принимать на основе качественных данных, потому что если мы на вход подадим некачественные данные, то решение наше будет соответствующего уровня. Управление данными осуществляется как активом (рис. 8), т. е. мы понимаем,

что данные являются частью цепочки создания стоимости, и, соответственно, мы управляем ими так же, как мы бы управляли станком, управляли бы недвижимостью, которую мы, например, сдаем в аренду.

Данные как объект управления

Рис. 8. Данные как объект управления

Технологии управления данными образуют технологии управления большими данными (рис. 9).

Технологии больших данных как объект управления

Рис. 9. Технологии управления данными

Все, что касается инструментов и средств управления данными, можно сказать, производства для работы с данными — это уже рынок больших данных.

Если говорить о прикладных инструментах, то для аналитики — это, безусловно, основа рынка больших данных, поскольку они развиваются достаточно интенсивно и позволяют решать все более сложные задачи.

Данные (рис. 8) и технологии (рис. 9) образуют приложения для решения конкретных задач, что является самым перспективным направлением, потому что задач становится все больше, и можно предлагать различные продукты из SIERVICE. Это огромный потенциал для развития рынка больших данных.

Все рассмотренные аспекты интересны не только для рынка больших данных. Это новое в администрировании БД ИС.

Выводы к главе 1

Рассмотрено влияние сквозных технологий цифровой экономики РФ на структуру и администрирование БД ИС.

Определены основные источники больших данных для ИС и показано, что для формирования данных необходима базовая инфраструктура — это оборудование для сбора данных и сеть для передачи данных, а также сервер, который обрабатывает эту информацию и, соответственно, производит запись.

Базовая инфраструктура — это наше оборудование, что должен понимать и представлять администратор ИС.

Рассмотрены две базовые функции: данные можно хранить и можно обрабатывать.

Далее по итогам обработки мы можем получить какой-то результат, и этот результат формируется технологическими инструментами, есть инструменты специальной работы с большими данными и там производится базовая аналитика, что меняет администрирование ИС.

Определено, что прикладные инструменты, базовые аналитики — это, безусловно основа рынка больших данных, поскольку они развиваются достаточно интенсивно и позволяют решать все более и более сложные задачи.

Рассматривая большие данные и управление ими, мы смотрим, чтобы данные были сформированы так, чтобы на их основе можно было принимать решение. Иначе смысла хранить данные практически нету.

Определяя инструменты и средства управления большими данными, можно сказать, производства для работы с данными — это уже рынок больших данных, что расширяет функции администрирования ИС.

Вопросы для самоконтроля

1. Является ли Big Data сквозной технологией цифровой экономики РФ?
2. Как изменили Big Data структуру и администрирование ИС?
3. Есть ли Big Data в банковских ИС?
4. Источники Big Data.
5. Как формируются большие данные?
6. Как изменилось администрирование в хранении и обработке в Big Data?
7. Цифровая инфраструктура, которая обеспечивает прием, хранение и обработку Big Data.
8. Большие данные как объект управления и администрирования.
9. Технологии управления большими данными.

ГЛАВА 2

Система управления базами данных для информационных систем в цифровой экономике

Администрирование ИС цифровой экономики, и прежде всего для сквозной цифровой технологии Big Data, определяется применяемой СУБД. В учебном пособии мы будем рассматривать СУБД Greenplum.

2.1. Основные сведения о СУБД Greenplum

СУБД Greenplum — open-source продукт, массивно-параллельная реляционная СУБД для хранилищ данных с гибкой горизонтальной масштабируемостью и столбцовым хранением данных на основе PostgreSQL. Благодаря своим архитектурным особенностям и мощному оптимизатору запросов, СУБД Greenplum отличается особой надежностью и высокой скоростью обработки SQL-запросов над большими объемами данных, поэтому эта MPP-СУБД широко применяется для аналитики Big Data в промышленных масштабах.

Эксплуатация системы началась в 2005 г. фирмой США «Greenplum». В 2018 г. компания РФ «Arenadata», разработчик первого отечественного дистрибутива Apache Hadoop (программный проект с открытым исходным кодом, предназначенный для эффективной обработки больших пакетов данных), выпустила собственную MPP-СУБД Arenadata DB на основе Greenplum, адаптировав ее для корпоративного использования.

До недавнего времени распространенной структурой систем для аналитической обработки больших объемов данных являлись SPM (symmetric multiprocessing), или системы симметричной мультипроцессорной архитектуры, которые показаны на рис. 10.

Рис. 10. SMP-архитектура

Главной особенностью систем с такой архитектурой является наличие общих физических ресурсов, которые разделяются между несколькими процессорами сравнимой производительности.

Память служит, в частности, для передачи сообщений между процессорами, и при этом все вычислительные устройства при обращении к памяти имеют равные права, поэтому структура называется симметричной.

Большинство популярных СУБД, таких как Oracle, Sybase и PostgreSQL, реализованы именно в такой архитектуре.

PostgreSQL принято обозначать 

SMP-системы обладают рядом преимуществ, таких как высокая скорость обмена данными между процессорами за счет наличия общей памяти, простота и универсальность обслуживания и относительно невысокая цена.

Существенным недостатком таких систем является плохая масштабируемость.

Каждый раз, когда необходимо увеличить скорость обработки данных, нам необходимо увеличивать мощность сервера за счет увеличения числа процессоров.

Как правило, данная операция является дорогостоящей, а в некоторых случаях — и вовсе невозможной из-за физических ограничений в конфигурации сервера.

Для решения проблемы масштабируемости в системах аналитической обработки данных стали применять архитектуру

MPP, или архитектуру массивно-параллельной обработки данных, показанную на рис. 11.

Рис. 11. MPP-архитектура

В такой архитектуре система состоит из нескольких независимых узлов, соединенных по сети. При этом в каждом вычислительном узле процессор обладает своими собственными физическими ресурсами, такими как память и диски, которые не разделяются с другими узлами. Именно поэтому такая архитектура также называется Shared nothing («Ничего общего»).

В MPP-системах вычислительная мощность и объем хранения данных увеличиваются за счет добавления в систему дополнительных вычислительных узлов.

Данный подход позволяет достигнуть линейного роста производительности в зависимости от количества узлов в системе.

СУБД Greenplum переназначена до хранения и обработки больших объемов данных методом распределения данных и обработки запросов на нескольких серверах.

Данная СУБД лучше всего подходит для построения корпоративных хранилищ данных, решения аналитических задач и задач машинного обучения и искусственного интеллекта.

В основе СУБД Greenplum лежит СУБД PostgreSQL. По сути СУБД Greenplum представляет из себя множество модифицированных экземпляров дисковых баз данных PostgreSQL, работаю-

щих совместно как одна связанная система управления базами данных.

В большинстве случаев СУБД Greenplum похож на PostgreSQL, например, в части синтаксиса SQL, функции параметров, конфигурации и функциональности для конечного пользователя. Пользователи базы данных взаимодействуют с СУБД Greenplum так же, как с обычной СУБД PostgreSQL.

Внутреннее устройство СУБД PostgreSQL было изменено или дополнено для поддержки параллельной структуры баз данных СУБД Greenplum.

Например, компоненты системного каталога, оптимизатора, исполнителя запросов и диспетчера транзакции были изменены и улучшены, чтобы они могли выполнять запросы одновременно в параллель во всех экземплярах базы данных.

СУБД Greenplum является исключительно программным решением. Он поставляется в виде дистрибутива, который может быть установлен на различные серверные платформы. Производительность напрямую зависит от оборудования, на котором он установлен.

СУБД Greenplum предоставляет широкий выбор инструментов для решения аналитических задач, показанных на рис. 12.

Рис. 12. Типы и формат данных, поддерживаемых СУБД Greenplum

СУБД Greenplum обладает большим количеством различных коннекторов, которые предоставляют доступ к другим источникам данных, таким как другие СУБД-компоненты экосистемы Hadoop и стриминговые источники данных, например, Kafka, показанные на рис. 13.

Рис. 13. Источник данных СУБД Greenplum

Hadoop — проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределенных программ, работающих на кластерах, из сотен и тысяч узлов.

Также в СУБД Greenplum можно разрабатывать хранимые процедуры не только на встроенном процедурном языке SQL, но и на многих других популярных языках программирования, которые будут компилироваться и выполняться внутри базы данных.

Также есть возможность разрабатывать процедуры, которые будут работать в контейнерах.

Такой подход позволяет сделать разработку безопасной путем изолирования исполняемого кода от операционной системы, на которой установлена СУБД. Языки программирования, поддерживаемые СУБД Greenplum, показаны на рис. 14.

Рис. 14. Языки программирования, поддерживаемые СУБД Greenplum

2.2. Архитектура СУБД Greenplum

СУБД Greenplum представляет из себя множество модифицированных баз данных PostgreSQL, которые расположены на нескольких серверах и взаимодействуют друг с другом, образуя единую, цельную базу данных, как показано на рис. 15.

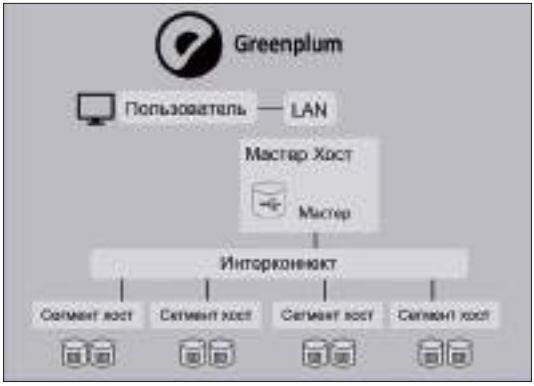

Рис. 15. Архитектура СУБД Greenplum

Логическая архитектура СУБД Greenplum выглядит следующим образом.

1. Мастер — это входная точка для СУБД Greenplum — это экземпляр базы данных СУБД PostgreSQL, к которому клиенты подключаются и отправляют свои SQL-запросы. Мастер координирует работу с другими экземплярами базы данных системы, которые называются сегментами. Пользователи взаимодействуют с мастером также, как если бы это была обычная база данных СУБД PostgreSQL. Подключение может выполняться при помощи клиентских программ, таких как:

— PSQL (SQL СУБД PostgreSQL);

— JDBC (Java DataBase Connectivity) — соединение с базами данных на Java;

— ODBC (Open Database Connectivity) — программный интерфейс (API) доступа к базам данных, разработанный компанией Microsoft.

На мастере располагается глобальный системный каталог — это набор системных таблиц, которые содержат метаданные о всех объектах базы данных СУБД Greenplum.

Важно отметить, что мастер не хранит какие-либо пользовательские данные, все пользовательские данные хранятся на сегментах. Функции мастера заключаются в следующем: мастер выполняет авторизацию клиентских соединений, обрабатывает входящие SQL-команды, распределяет нагрузку между сегментами, координирует результаты, возвращаемые каждым сегментом, выполняет финальные операции над данными, предоставляет конечный результат клиенту.

2. Сегменты — это отдельные экземпляры базы данных СУБД PostgreSQL, которые хранят определенную порцию пользовательских данных и выполняют большую часть обработки запросов.

Когда пользователь подключается к базе данных через мастер и запускает запрос, на каждом сегменте создаются процессы для обработки этого запроса. Затем каждый сегмент параллельно обрабатывает свою порцию данных и возвращает финальный результат своей работы на мастер — сегменты работают на серверах, которые называются сегмент-хосты, или ноды.

На каждом ноде обычно располагается от двух до восьми сегментов Greenplum. Количество сегментов конфигурируется при первоначальной установке системы и напрямую зависит от профиля рабочих нагрузок системы и технических характеристик сервера, таких как количество и производительность ядер процессора, объема оперативной и постоянной памяти и сетевых интерфейсов.

Предполагается, что все ноды системы будут настроены идентично. Ключом к достижению максимальной производительности от СУБД Greenplum является равномерное распределение данных и рабочих нагрузок по большому количеству сегментов с одинаковой производительностью. Это необходимо для того, чтобы все сегменты одновременно начинали работать над задачей и одновременно завершали свою работу. В этом случае ни один из узлов не станет узким горлышком всей системы.

3. Интерконнект — это сетевой слой архитектуры СУБД Greenplum. Интерконнект обеспечивает взаимосвязь между мастером-сегментами и сетевой инфраструктурой, в которой развернут кластер.

Для обеспечения высокой производительности системы рекомендуемая пропускная способность сети должна быть не менее 10.0 гигабит.

По умолчанию обмен данными происходит по протоколу UDP (User Datagram Protocol — протокол пользовательских блоков информации; один из ключевых элементов набора сетевых протоколов для Интернета).

Также программное обеспечение СУБД Greenplum выполняет дополнительную проверку пакетов поверх UDP. Таким образом, надежность остается такой же, как у TCP (Transmission Control Protocol — один из основных протоколов передачи данных Интернета). При этом производительность и возможности масштабирования намного выше.

2.3. Дистрибьюция

Дистрибьюция, или распределение данных — это один из важнейших факторов быстродействия системы. В СУБД Greenplum общее время выполнения запроса измеряется вре-

менем выполнения на всех сегментах. Так как обработка на сегментах выполняется в параллель, то максимальная скорость системы ограничивается работой самого медленного сегмента. Если данные распределены неравномерно, сегменты с большим количеством данных будут выполнять работу дольше, что снижает общую производительность. Таким образом, для повышения производительности необходимо стремиться к равномерному распределению данных по всем сегментам системы (рис. 16).

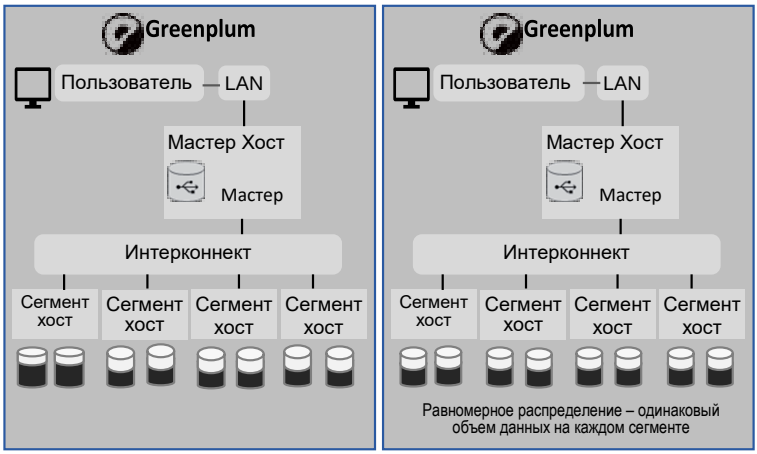

Рис. 16. Неравномерное и равномерное распределение данных по сегментам

Понимание и правильное использование такой ключевой особенности СУБД Greenplum, как распределение данных по всем сегментам с учетом особенностей ваших данных, поможет вам добиться наилучшей производительности. Даже одна таблица в СУБД Greenplum хранится не на одном, а на нескольких сегментах. Каждый сегмент хранит уникальную порцию данных. Для каждой таблицы задается своя политика дистрибьюции. Рассмотри пример, представленный на рис. 17.

Способ, согласно которому строки распределяются по сегментам, задается при создании таблицы в команде Create Table, а также может быть изменен впоследствии командой Alter Table.

Рис. 17. Таблица с распределением

Ключ дистрибьюции — это поле или набор полей, по значениям которых определяется, на каком сегменте будет храниться существующая строка таблицы.

При выборе ключа дистрибьюции необходимо учитывать два основных момента:

— ключ должен обеспечивать равномерность распределения данных;

— нужно добиться локальности операций.

Под локальными операциями подразумеваются ситуации, когда соединяемые таблицы имеют равный ключ дистрибьюции, что позволяет выполнять соединение локально на сегментах без необходимости перераспределять данные, что, в свою очередь, дает существенное ускорение выполнения запроса СУБД Greenplum.

Существует три вида дистрибьюции:

— distributed by;

— distributed randomly;

— distributed replicated.

Distributed by — это дистрибьюция по hash указанных полей. Идея соединения с помощью хеширования состоит в поиске подходящих строк с помощью заранее подготовленной хеш-таблицы.

Хеш-таблица позволяет сохранять пары, составленные из ключа хеширования и значения, а затем искать значения

по ключу за фиксированное время, не зависящее от размера хеш-таблицы.

В нашем примере из значения колонки или нескольких колонок вычисляется хеш-значение, по которому в дальнейшем определяется, на какой сегмент попадает запись. В нашем примере вычисление хеш-значения осуществляется по значениям одной или двух колонок.

Distributed randomly — это метод, при котором планировщик сам равномерно размещает данные по всем сегментам. Данный метод лучше всего подходит для таблиц, в которых отсутствует или сложно определить ключ, который обеспечит равномерное распределение данных. Важно учесть, что при таком алгоритме соединение с другими таблицами будет всегда вызывать перераспределение данных, так как строки распределены случайным образом.

Distributed replicated — это метод, при котором на каждом сегменте хранится полная копия таблицы.

Такой метод подходит для небольших таблиц, например, справочников, которые регулярно участвуют в соединениях с другими таблицами.

Хранение полной копии таблицы на каждом сегменте позволяет сэкономить время, которое в противном случае было бы потрачено на перераспределение во время выполнения запроса.

При выборе ключа дистрибьюции необходимо руководствоваться следующими правилами:

— в качестве ключа дистрибьюции необходимо выбирать поле с большой селективностью, т. е. поля с большим количеством уникальных значений. Как правило, под этот критерий подходят поля с идентификаторами. Если данные распределены по ключу с низкой селективностью, то велика вероятность неравномерного распределения данных, так как одинаковые значения будут располагаться на одном сегменте;

— в ключе не должно быть Null или значений по умолчанию. Иначе все Null или другие значения по умолчанию будут попадать на один сегмент, что приведет к перекосу данных.

Вернемся к рассмотрению примера, представленного на рис. 17.

Выберем в качестве ключа дистрибьюции поле ID. В результате распределения на каждый сегмент падает ровно по две строки. Распределение получается равномерное, а значит, ключ дистрибьюции хороший.

Выберем в качестве ключа дистрибьюции поле group. На рис. 18 показаны результаты.

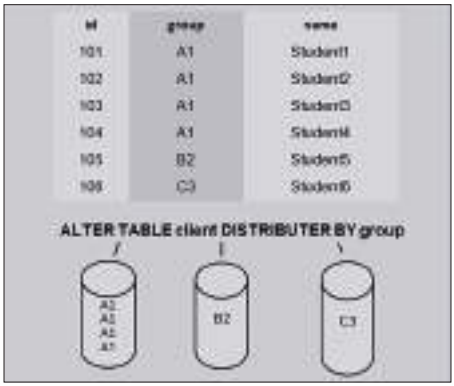

Рис. 18. Ключ дистрибьюции — поле group

После распределения мы видим, что на одном из сегментов данных в 4 раза больше, чем на других. Как следствие, и работать этот сегмент будет в 4 раза дольше. Делаем вывод о том, что ключ выбран неоптимально. Проверить равномерность распределения строк между сегментами можно, используя служебный столбец GP сегмент ID, который присутствует в каждой таблице:

```
SELECT gp_segment_id, count (1)
FROM sales
GROUP BY gp.segmentjd
```

В данном столбце содержится идентификатор сегмента, на котором лежит строка «Пример запроса» с группировкой по полю GP, сегмент ID для определения равномерности распределения строк по сегментам, он приведен на экране. По результату запроса видно, что распределение данных таблицы Sails практически равномерное (рис. 19).

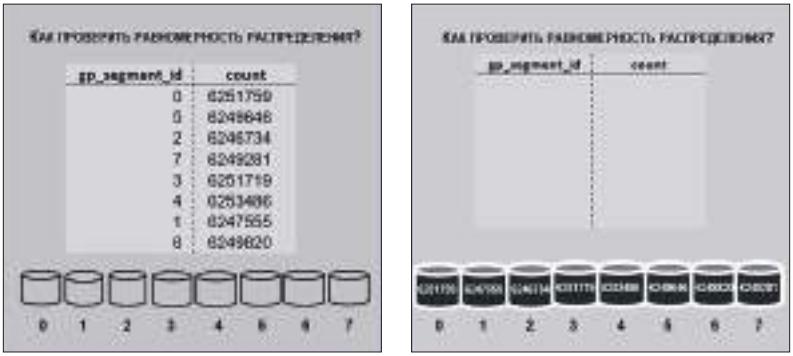

Рис. 19. Равномерное распределение данных таблицы

Ключ распределения можно изменить без пересоздания таблицы с помощью предложения:

ALTER TABLE tabi SET DISTRIBUTED BY (coll)

ALTER TABLE tab! SET DISTRIBUTED RANDOMLY

Также есть возможность перераспределения данных по прежнему ключу дистрибьюции при помощи опции:

ALTER TABLE tabi SET WITH (REORGANIZE=true)

Эта опция будет полезна в двух случаях: при добавлении нового сегмента и для избежания раздутия таблиц, результат показан на рис. 20.

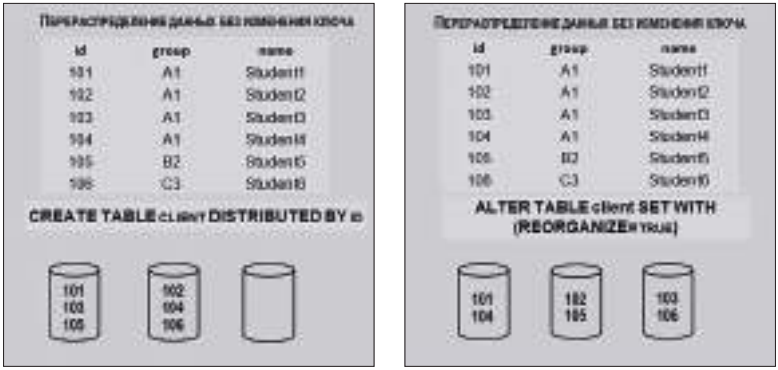

Рис. 20. Перераспределение данных без изменения ключа

При добавлении нового сегмента хоста в кластер возникает необходимость заново перераспределить данные, чтобы избежать перекоса в распределении данных. Когда новый сегмент фактически не задействован для хранения данных, а все данные по-прежнему распределены по старым сегментам.

Некоторые важные рекомендации, которые помогут вам правильно распределять данные:

— явно задавайте способ и ключ распределения при создании таблицы, в том числе для временных таблиц;

— используйте RANDOMLY-распределение для небольших таблиц, где нет хороших кандидатов на ключ распределения из одного атрибута;

— избегайте использования в качестве ключа распределения атрибутов, которые вы будете часто указывать в предложении WHERE и атрибутов с типом данных date, timestamp;

— старайтесь использовать в качестве ключа распределения тип данных integer (предпочтительнее, чем string) и столбец, который будет использоваться как ключ соединения (JOIN ON).

2.4. Партиционирование

Партиционирование, или секционирование, представляет собой логическое разделение таблицы на части по определенному критерию.

Применение партиционирования на больших таблицах значительно улучшает производительность запросов, а также упрощает сопровождение базы данных благодаря возможности разделять перемещать и компактно хранить данные в зависимости от их востребованности.

Партиционированные таблицы, также как и любые таблицы в СУБД Greenplum, имеют ключ дистрибьюции.

Важно различать понятия:

— **дистрибьюция** — это распределение данных по сегментам с целью равномерно распределить нагрузку по всем узлам кластер;

— **партиционирование** — это метод оптимизации, при котором на каждом сегменте таблица делится на части с целью

увеличения производительности запросов за счет выборочного сканирования.

Различия дистрибьюции и партиционирования показаны на рис. 21.

Рис. 21. Различия дистрибьюции и партиционирования

Партиционирование распространяется на все сегменты и выполняется на каждом сегменте одинаково.

Например, если таблица при создании поделена на 6 партиций, то на каждом сегменте кластера будет по 6 партиций.

Когда таблица партиционирована, запрос выполняется гораздо быстрее за счет того, что на каждом сегменте сканируется только часть таблицы.

Дистрибьюция в СУБД Greenplum обязательна для всех таблиц, т. е. все таблицы в СУБД Greenplum являются распределенными и имеют ключ дистрибьюции.

Партиционирование — это лишь опция, которую можно применять к таблице в целях оптимизации.

В СУБД Greenplum поддерживаются следующие виды позиционирования:

— PARTITION BY RANGE — по диапазону значений;

— PARTITION BY LIST — по списку значений;

— PARTITION BY ... SUBPARTITION BY — многоуровневое.

Рассмотрим таблицу, содержащую записи о клиентах учебного заведения, например, студентах университета (рис. 22).

Согласно заданной политике дистрибьюции, таблица равномерно распределена по сегментам на основе идентификаторов студентов.

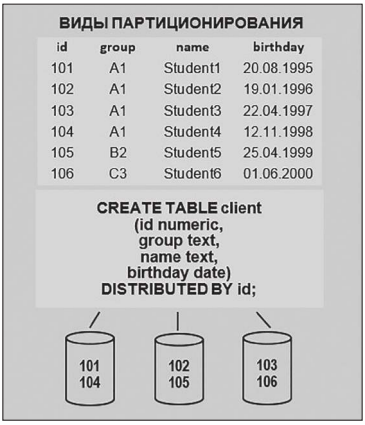

Рис. 22. Создание таблицы «Студенты университета»

Допустим, что для построения отчетности периодически требуется анализировать информацию о всех учащихся того или иного возраста. Вместе с тем мы хотим, чтобы система выдавала результаты запросов для разных возрастов как можно быстрее. Эту задачу поможет решить Range. Партиционирование по году рождения приведено на рис. 23.

Рис. 23. Партиционирование по году рождения

Вид партиционирования задается при создании таблицы с помощью выражения Partition By Range. Диапазон значений за-

дан выражением **start** и **end**, а шаг каждой партиции — выражением **every interval.** После создания таблицы на каждом сегменте будут созданы по 6 партиций, хранящих записи по году.

Если в запросе к таблице есть условия фильтрации записи по году, то база данных в рамках каждого сегмента сразу обращается к той партиции, которая соответствует нужному году, в связи с чем запрос выполняется намного быстрее (рис. 24).

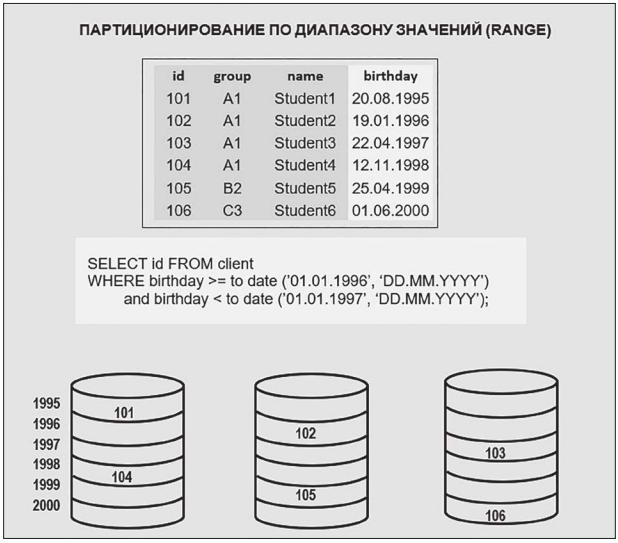

Рис. 24. Партиционирование по диапазону значений

В случае партиционирования типа List разделение данных выполняется на основе списка значений.

Например, мы можем применить к той же таблице партиционирование по полю «группа студентов» (рис. 25).

Разделение партиции и по значениям выполняется с помощью выражения Partition Values.

На каждом сегменте создается по 3 партиции — для групп A1, B2 и C3. Внутри каждой партиции есть студенты с разными годами рождения. Например, запись о студенте 4 хранится в сегменте 104 в партиции A1.

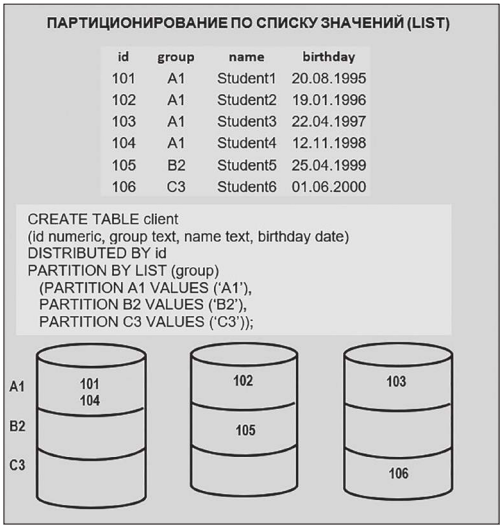

Рис. 25. Партиционирование по списку значений

Этот способ партиционирования будет удобен, если требуется часто строить запросы с фильтрацией по группе студентов, кроме того, в одной таблице можно реализовать многоуровневое партиционирование, при котором будут использоваться комбинации обоих типов. Проводится по группе студентов, а внутри группы — по году рождения (рис. 26).

Имейте в виду, что использование многоуровневого партиционирования может привести к созданию избыточного количества партиций, что, в свою очередь, может замедлить работу всей базы данных.

При создании партиционированной таблицы можно указывать:

— отличающиеся имена;

— отличающиеся диапазоны;

— разные опции хранения (ориентация, компрессия);

— можно создать партицию по умолчанию (DEFAULT) для значений, не удовлетворяющих условиям других партиций.

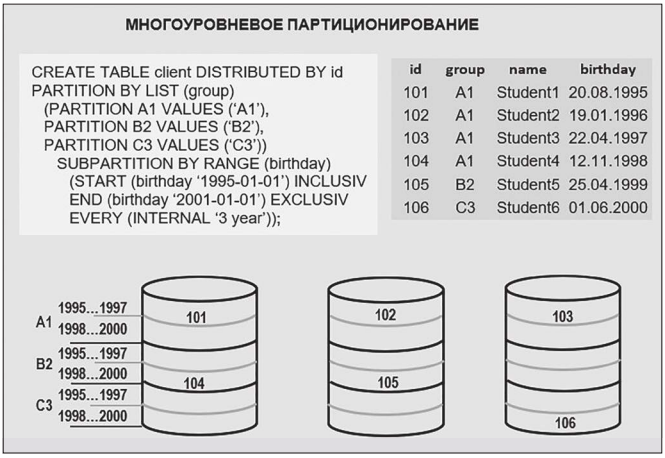

Рис. 26. Многоуровневое партиционирование

Нельзя:

— указать разные ключи дистрибьюции для партиций одной таблицы;

— поменять тип партиционирования без пересоздания таблицы.

Рассмотрим пример стратегии партиционирования с применением разных опций хранения данных (рис. 27).

Рис. 27. Стратегия партиционирования с применением разных опций хранения данных

Наиболее свежие данные хранятся без компрессии для того, чтобы можно было быстро выполнять операции обновления.

Старые данные хранятся с более сильными алгоритмами сжатия. Например, для данных трехмесячной давности применяется низкая степень сжатия, что обеспечивает хорошую производительность, так как выполняется мало операций ввода-вывода.

Для данных годичной давности применяется колоночная ориентация с более высокой степенью компрессии, что повышает производительность при выборке по подмножеству колонок.

Для данных глубиной 2 года используется колоночная ориентация и максимальная компрессия. При этом производительность ниже, так как при выполнении запросов строки приходится разжимать.

Архивные данные хранятся за пределами базы данных в виде файлов.

Обращаем внимание, что цель партиционирования состоит в исключении чтения лишних партиций, т. е. минимизации количества сканируемых партиций при выполнении запроса.

Следовательно, наиболее эффективным будет такое позиционирование, которое максимально приближено к логике выполнения запроса.

Последовательность операции при выполнении запроса отражается в плане запроса, который выводится с помощью команды Explain.

Рассмотрим план выполнения запросов в случае таблицы с партиционированием или без него (рис. 28).

В плане запроса для таблицы без партиционирования сканируется вся таблица, поэтому стоимость выполнения запроса получается высокой. Для таблицы с партиционированием мы видим, что сканируется только одна нужная партиция, в связи с чем стоимость запроса низкая, и, как следствие, запрос выполняется намного быстрее.

Рассмотрим синтаксис запросов — для разных партиций можно указывать разные интервалы и опции хранения.

Рис. 28. План выполнения запросов при партиционировании

На рис. 29 для партиций с интервалом в один год или 3 месяца задана колоночная ориентация хранения, и тип компрессии задан с разными коэффициентами сжатия.

СИНТАКСИС. ИНТЕРВАЛЫ ХРАНЕНИЯ ДЛЯ ПАРТИЦИЙ

```
CREATE TABLE orders
(order_id BIGINT,
order_date TIMESTAMP WHIT ZONE,
order_mode VARCHAR(8),
customer_id INTEGER)
DISTRIBUTED BY (customer_id)
PARTITION BY RANGE (order_date)
    (start (date '2008-01-01') end (date '2011-01-01') every (interval '1 year')
        with (appendonly=true, orientation=column, compresstype=zlib, compresslevel=3),
    start (date '2011-01-01') end (date '2013-01-01') every (interval '3 months')
        with (appendonly=true, orientation=column, compresstype=zlib, compresslevel=1),
    start (date '2013-01-01') end (date '2014-01-01') every (interval '1 months')
        with (appendonly=true, orientation=row, compresstype=zstd),
    start (date '2014-01-01') end (date '2014-02-01') every (interval '1 day'));
```

Рис. 29. Синтаксис запроса

Для партиций с интервалом в один месяц применяется ориентация по строкам и другой тип компрессии без явного указания коэффициента сжатия. Для партиций с интервалом в один день используется способ хранения по умолчанию.

Рассмотрим синтаксис изменения партиции (рис. 30).

Рис. 30. Добавление партиции

Для изменения партиции используется команда **Alter table** с дополнительными опциями.

```
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO jan16;
```

1. Очистка партиции.

```
ALTER TABLE table.prtmixed TRUNCATE PARTITION year2019;
```

2. Очистка партиции.

```
ALTER TABLE tabLe.prt.mixed TRUNCATE PARTITION year2019;
ALTER TABLE tabLe.prt.mixed TRUNCATE PARTITION FOR ('2019-04-
28'::date);
```

Для очистки партиции и применяется команда Trancate Partition. В примере показано, как очистить партицию с названием «яр 2019.0» или партицию, содержащую дату 28.0 апреля 2019.0 года.

3. Удаление партиции.

```
ALTER TABLE cardUst DROP PARTITION n.a;
ALTER TABLE card.list DROP DEFAULT PARTITION;
```

4. Удаление партиции.

```
ALTER TABLE cardUst DROP PARTITION n.a IF EXIST;
ALTER TABLE card.list DROP DEFAULT PARTITION IF EXIST;
```

Для удаления партиции используется команда DROP PARTITION.

В примере показано удаление обычной и дефолтной партии. При желании в тексте команды можно указать опцию IF EXIST, которая поможет избежать сообщения об ошибке при выполнении команды в случае отсутствия удаляемой партиции. Для изменения опции хранения в отдельной партии можно создать таблицу с нужной опцией хранения, вставить в нее данные из этой партиции и затем заменить всю прежнюю партицию на только что созданную таблицу с помощью команды ALTER TABLE.

5. Разделение партиции.

```
ALTER TABLE sales SPLIT PARTITION FOR ('2021-01-01') AT
('2021-01-16')
INTO (PARTITION jan21.1to15, PARTITION jan21.16to31);
ALTER TABLE sales SPLIT DEFAULT PARTITION START ('2021-01-
01') INCLUSIVE
END ('2021-02-01') EXCLUSIVE
INTO (PARTITION jan21, default partition);
CREATE TABLE sales (trans.id int, date date, amount decimal
(9,2), region text) DISTRIBUTED BY (trans.id) PARTITION BY
RANGE (date) SUBPARTITION BY LIST (region) SUBPARTITION
TEMPLATE
(SUBPARTITION usa VALUES ('usa'), SUBPARTITION asia VALUES
('asia'), SUBPARTITION europe VALUES ('europe'), DEFAULT
SUBPARTITION other.regions) ( START (date '2014-01-01')
INCLUSIVE END (date '2014-04-01') EXCLUSIVE EVERY (INTERVAL
'1 month'));
```

Для разделения одной партиции на две части используется команда ALTER TABLE … SPLIT PARTITION. В начале примера

выполняется разделение партиции, в которой хранятся данные за январь, на две партии. В первой будут храниться данные с 1 по 15 января, а во второй — с 16 по 31; если в вашей таблице есть дефолт-партиции, то добавить новую партицию в такой таблице можно только путем разделения дефолт-партиции, как это показано во второй части. В этом случае в предложении INTO в качестве второй позиции необходимо указать дефолт-партицию.

Примеры операций Data Definition Language (DDL).

```
ALTER TABLE sales ADD PARTITION "4"
START ('2014-04-01') INCLUSIVE
END ('2014-05-01') EXCLUSIVE ;
```

При создании таблицы возможно определение шаблона, по которому будут автоматически создаваться наборы субпартиций при добавлении новых партиций.

Создание и изменение шаблонов субпартиций.

```
ALTER TABLE sales ADD PARTITION "4"
START ('2014-04-01') INCLUSIVE
END ('2014-05-01') EXCLUSIVE ;
ALTER TABLE sales SET SUBPARTITION TEMPLATE ( SUBPARTITION
usa VALUES ('usa'), SUBPARTITION asia VALUES ('asia'),
SUBPARTITION europe VALUES ('europe'), SUBPARTITION africa
VALUES ('africa'), DEFAULT SUBPARTITION regions);
```

В приведенном скрипте создания шаблон определяется с помощью предложения SUBPARTITION TEMPLATE, в котором мы определяем 4 субпартиции для разных регионов. Теперь, если мы добавляем в таблицу партицию и следующим скриптом, то эта партиция по шаблону автоматически разделится на 4 субпартиции и для всех регионов.

Просмотр информации о партициях.

Информация о ключах партиционирования:

```
SELECT * FROM pg_catalog.pg_partitions WHERE
tablename = 'sales';
```

С помощью представления о pg Partitions можно получить подробную информацию о партиционированных таблицах и их иерархии. Представление pg Partition Colance отображает информацию о колонках, которые используются для партиционирования.

Когда использовать.

Большая таблица.

Фильтры запросов (WHERE) исключают часть партиций.

Нужно «скользящее окно» данных при периодической загрузке архивировании.

Как использовать.

Нужно:

1) предпочитать RANGE, а не LIST;

2) партиционировать по часто используемому столбцу;

3) убедиться в исключении ненужных партиций в плане запроса (EXPLAIN);

4) выбирать наилучший способ физического хранения для разных партиций (например, по строкам/колонкам).

Не рекомендуется:

1) создавать большое количество партиций;

2) использовать дефолтную партицию;

3) использовать многоуровневое партиционирование;

4) секционировать по столбцу ключа дистрибьюции;

5) создавать много партиций при колоночной ориентации (количество физических файлов = сегменты × столбцы × партиции).

Выводы к главе 2

Традиционно администрирование ИС взаимосвязано с администрированием БД и ее СУБД.

Сквозные технологии цифровой экономики РФ Big Date и Blockchain изменили структуру и функции ИС, существенно изменили функции администрирования данных и программ их обработки.

Рассмотрен новый тип СУБД для Big Data. СУБД Greenplum — open-source продукт, массивно-параллельная реляционная СУБД для хранилищ данных с гибкой горизонтальной масштабируемостью и столбцовым хранением данных на основе PostgreSQL.

Благодаря своим архитектурным особенностям и мощному оптимизатору запросов, СУБД Greenplum отличается особой надежностью и высокой скоростью обработки SQL-запросов над большими объемами данных, поэтому эта MPP-СУБД широко применяется для аналитики Big Data в промышленных масштабах.

Показано, что основная идея организации хранения и обработки Big Data — распараллеливание хранения и обработки данных.

Эти особенности становятся определяющими в администрировании подобных ИС.

Показано преимущество для решения проблемы масштабируемости в ИС аналитической обработки больших данных архитектуры MPP (massive parallel processing), или архитектуры массивно-параллельной обработки данных. В такой архитектуре система состоит из нескольких независимых узлов, соединенных по сети. При этом в каждом вычислительном узле процессор обладает своими собственными физическими ресурсами, такими как память и диски, которые не разделяются с другими узлами, что должно являться основой в администрировании.

В учебном пособии рассматривается СУБД Greenplum, предназначенная для хранения и обработки больших объемов данных методом распределения данных и обработки запросов на нескольких серверах.

Данная СУБД лучше всего подходит для построения корпоративных хранилищ данных, решения аналитических задач и задач машинного обучения и искусственного интеллекта.

Показано, что СУБД Greenplum представляет из себя множество модифицированных экземпляров дисковых баз данных PostgreSQL, работающих совместно как одна связанная система управления базами данных.

Определено, что СУБД Greenplum предоставляет широкий выбор инструментов для решения аналитических задач.

Основным средством управления данными является встроенный процедурный язык SQL, но можно применять и другие популярные языки программирования, которые будут компилироваться и выполняться внутри базы данных.

Показана возможность разрабатывать процедуры, которые будут работать в контейнерах. Такой подход позволяет сделать

разработку безопасной путем изолирования исполняемого кода от операционной системы, на которой установлена СУБД.

Подробно рассмотрена архитектура СУБД Greenplum.

Изучена дистрибьюция или распределении данных, что является одним из важнейших факторов быстродействия ИС.

Рассмотрено партиционирование, или секционирование, как логическое разделение таблицы на части по определенному критерию. Применение партиционирования на больших таблицах значительно улучшает производительность запросов, а также упрощает сопровождение базы данных благодаря возможности разделять, перемещать и компактно хранить данные в зависимости от их востребованности.

Рассмотрены примеры DDL-операций.

Вопросы для самоконтроля

1. Необходимо ли администрирование БД ИС? Как на это влияет коллективное использование БД?
2. Что изменится в администрировании данными БД Big Data?
3. В чем отличие СУБД Greenplum от традиционных?
4. Различие структуры системы для аналитической обработки больших объемов данных SMP от MPP.
5. Назовите общие характеристики СУБД Greenplum и в чем их преимущества перед традиционными СУБД.
6. Типы и форматы данных СУБД Greenplum.
7. Языки программирования, поддерживаемые СУБД Greenplum.
8. Архитектура СУБД Greenplum.
9. Что такое дистрибьюция и есть ли такая функция в традиционных СУБД?
10. Возможности партиционирования данных.
11. Где хранятся пользовательские данные в СУБД Greenplum?
12. К чему подключается клиент при начале работы с СУБД Greenplum?
13. Где в СУБД Greenplum выполняются запросы пользователей?

14. Что определяет пропускную способность канала взаимодействия сегментов СУБД Greenplum?

15. Какие задачи в СУБД Greenplum решает Мастер?

16. Дистрибьюция в Greenplum — это...?

17. Партиционирование в Greenplum — это...?

18. Что позволяет решить использование индексов в Greenplum?

19. Что относится к особенностям представлений (VIEW) как объектов БД?

20. Структура запроса SELECT. Типы соединений. LIMIT и OFFSET.

21. Определение и синтаксис оконной функции.

22. Что такое план запроса?

23. Улучшение производительности при выполнении запроса.

ГЛАВА 3

Определения хостинга для баз данных и информационных систем

Ключевую роль в выполнении задач администрирования БД ИС играет определение базовых параметров БД на сервере размещения ИС.

Дополнительно рассмотрим вопросы администрирования и ИС на примере сайта.

3.1. Подключение к серверу

Для того чтобы на сервере разместить БД и ИС (на примере сайта), необходимо выделить место на внешнем диске сервера и административные параметры. Будем использовать удаленный рабочий стол (RDR — Remote Desktop Protocol) для подключения к серверу (IP сервера и пароль — выдуманные).

На своем компьютере запускаем удаленный рабочий стол. Необходимо подключиться с использованием RDP к серверу. Появляется форма RDR (рис. 31).

Указываем IP-адрес сервера и имя пользователя. Нажимаем кнопку «Подключиться», появляется окно с сообщением об ошибках сертификата (рис. 32).

Необходимо нажать кнопку «Да» для установления соединения с сервером.

Если все сделано верно, отобразится рабочий стол сервера. Нам потребуется инструментарий, закрепленный на панели задач «Инструментарий сервера для администрирования»: проводник, диспетчер сервера, диспетчер SQL.

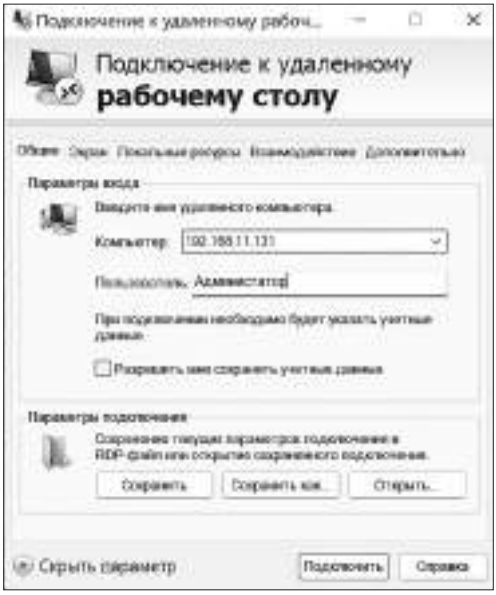

Рис. 31. Форма RDR

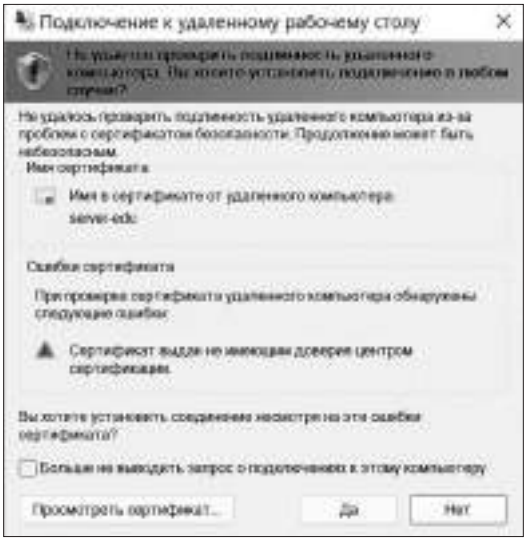

Рис. 32. Окно об ошибке

3.2. Создание каталога на диске сервера с файловым содержимым сайта

Каталоги сайтов располагаются на сервере по адресу: «C:\inetpub\wwwroot\sites\LocalUser\имя_пользователя».

Создадим каталог средствами проводника Windows — папку хостинга с именем newhost (рис. 33).

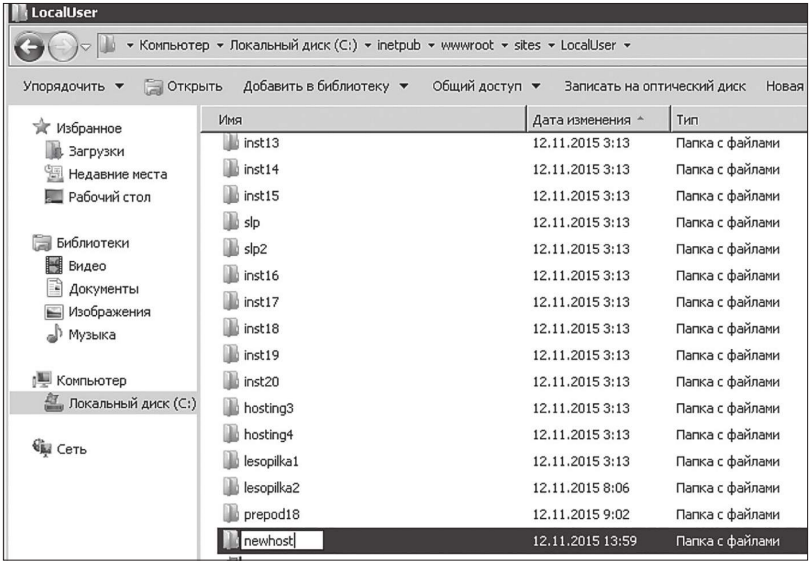

Рис. 33. Создание каталога на диске сервера

Самым распространенным доступом к хостингу является метод ftp (File Transfer Protocol) — это протокол доступа, предназначенный для удаленной передачи файлов.

3.3. Создание пользователя для доступа по ftp

Необходимо открыть оснастку «Диспетчер сервера» щелчком по кнопке на панели задач, открыть куст «Диспетчер сервера» → Конфигурация → Локальные пользователи и группы → Пользователи.

Затем кликнуть правой кнопкой мыши по значку «Пользователи» и в появившемся контекстном меню выбрать пункт «Новый пользователь» (рис. 34).

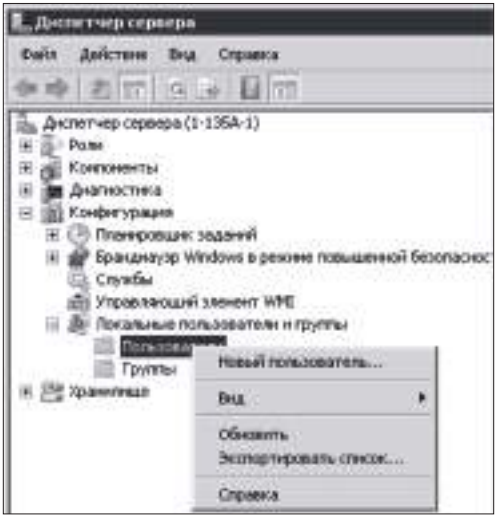

Рис. 34. Новый пользователь

В появившемся окне необходимо поставить знаки в полях выбора, как указано на скриншоте ниже (срок действия пароля неограничен), снять с «Требовать соблюдения политики паролей», ввести имя пользователя и пароль (рис. 35).

Имя пользователя должно совпадать с именем каталога, созданном на шаге 1. В случае несоблюдения этого правила доступ по FTP будет невозможен!

Убедившись в верном заполнении всех полей, необходимо нажать кнопку «Создать», затем — «Закрыть».

В открытом кусте «Пользователи» оснастки «Диспетчер сервера» (рис. 36) необходимо отыскать недавно созданного пользователя, вызвать контекстное меню щелчком правой кнопкой мыши по имени пользователя и выбрать пункт «Свойства» контекстного меню.

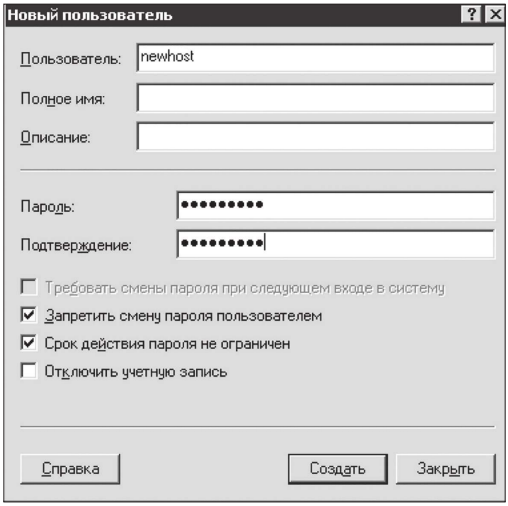

Рис. 35. Создание пароля

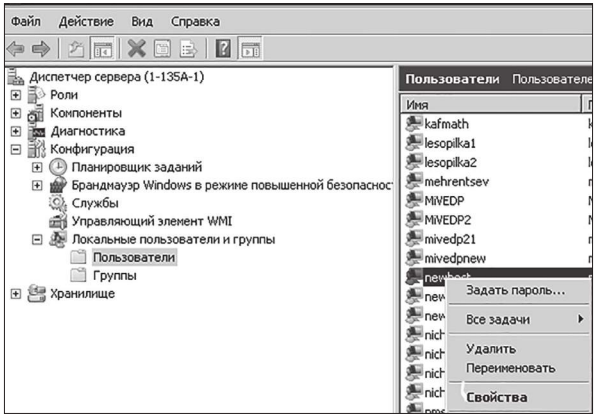

Рис. 36. Параметризация пользователя

В открывшемся окне «Свойства пользователя» (рис. 37) на начальной вкладке имеется возможность исправить параметры пароля в случае неверно установленных знаков на этапе создания пользователя.

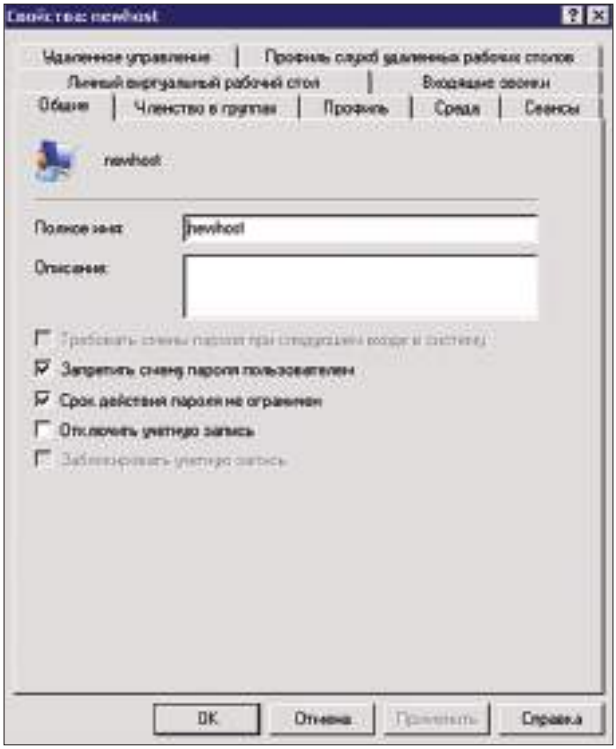

Рис. 37. Свойства пользователя

Далее необходимо открыть вкладку «Членство в группах», в поле «Член групп» выделить имеющуюся группу «Пользователи» и нажать кнопку «Удалить». В результате поле «Член групп» должно стать пустым (рис. 38).

Для того чтобы дать пользователю права на доступ по ftp к каталогам сайта, необходимо нажать кнопку «Добавить» и в открывшемся модальном окне «Выбор: Группы» в поле «Введите имена выбираемых объектов» ввести имя «SiteUsers» и нажать кнопку OK (рис. 39).

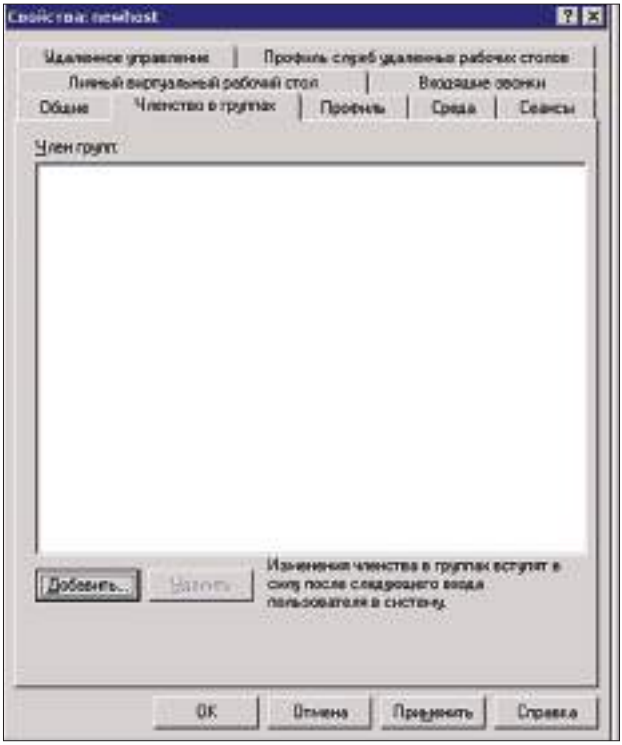

Рис. 38. Членство в группах

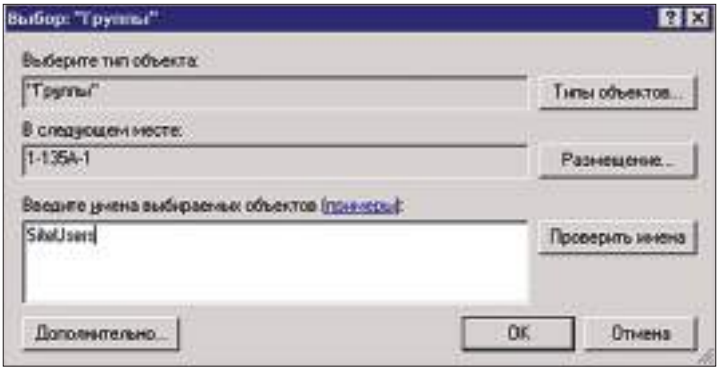

Рис. 39. Выбор «Группы»

Если ошибок в имени не допущено, модальное окно закроется без сообщений и активируется окно «Свойства пользователя», где в окне «Члены групп» будет визуализироваться группа «SiteUsers» (рис. 40).

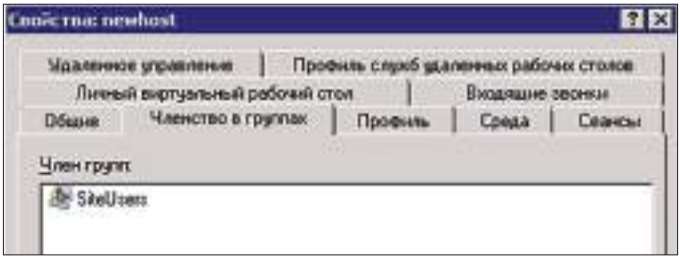

Рис. 40. Членство в группах

Убедившись в наличии в поле ввода «Член групп» единственной записи «SiteUsers», необходимо нажать кнопку OK.

3.4. Создание пользователя базы данных

Манипуляции с БД осуществляются с использованием инструмента «Среда Microsoft SQL Server Management Studio». Запуск среды осуществляется щелчком по кнопке «Диспетчер SQL» на панели задач. Откроется окно настройки, где системой предлагается выбрать машину с запущенным SQL-сервером, а также параметры аутентификации (рис. 41).

Для поставленных целей подходят значения сервера по умолчанию, поэтому ничего не меняя, мы нажимаем кнопку «Соединить». Откроется рабочее окно среды (рис. 42).

Для создания пользователя базы данных необходимо раскрыть каталог «Безопасность → Имена входа» и в контекстном меню папка «Имена выхода» выбрать пункт «Создать имя входа» (рис. 43).

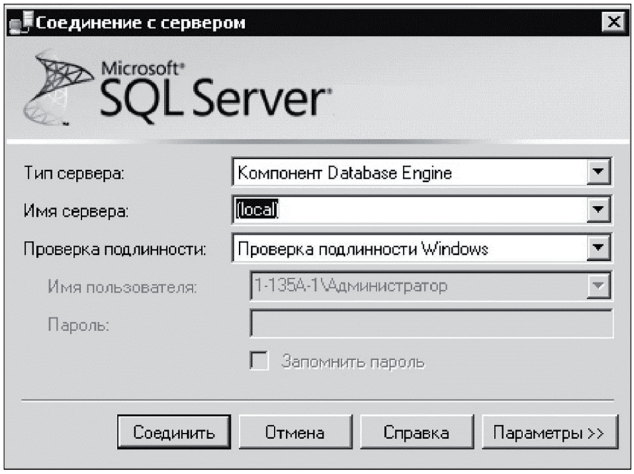

Рис. 41. Соединение с сервером БД SQL

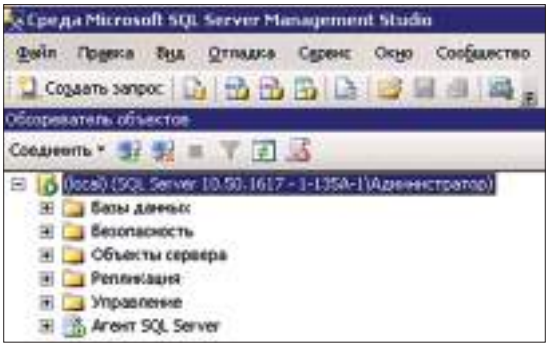

Рис. 42. Рабочее окно среды SQL

Рис. 43. Создание имени входа БД

Откроется окно (рис. 44).

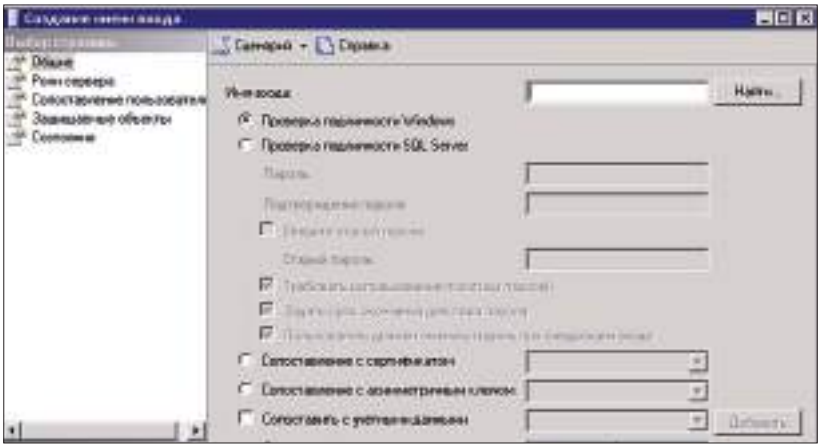

Рис. 44. Создание имени входа БД

В поле «Имя входа» вводим имя входа, которое потом будет в дальнейшем использоваться как логин при формировании строки подключения к БД в приложении ИС.

Затем необходимо переключить радиокнопку с позиции «Проверка подлинности Windows» на «Проверка подлинности SQL Server», и в ставших доступными полях «Пароль» и «Подтверждение пароля» ввести пароль, который будет использоваться при формировании строки подключения.

Также необходимо снять отметку с позиции «Требовать использование политики паролей» (рис. 45).

После нажатия на кнопку OK, имя входа будет создано. Оно используется для аутентификации при подключении к базе данных.

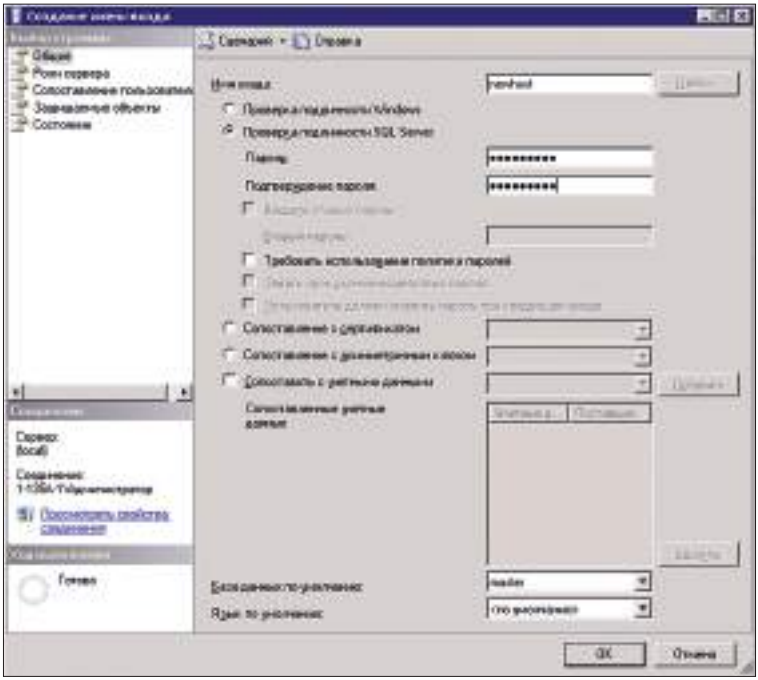

Рис. 45. Создание имени входа БД SQL

3.5. Создание базы данных

С помощью инструмента «Среда Microsoft SQL Server management Studio» (раздел 3.3) кликаем правой кнопкой мыши по папке «Базы данных» и в появившемся контекстном меню выбираем пункт «Создать базу данных».

Вводим желаемое имя базы данных в соответствующее поле «Владелец» (рис. 46).

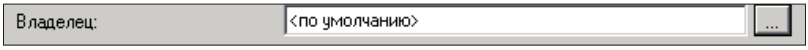

Рис. 46. Определение имени БД

Далее (рис. 46) нажимаем кнопку «Еще» и в открывшемся окне «Выбор владельца базы данных» вводим имя входа, созданное на предыдущем шаге (рис. 47), и нажимаем OK.

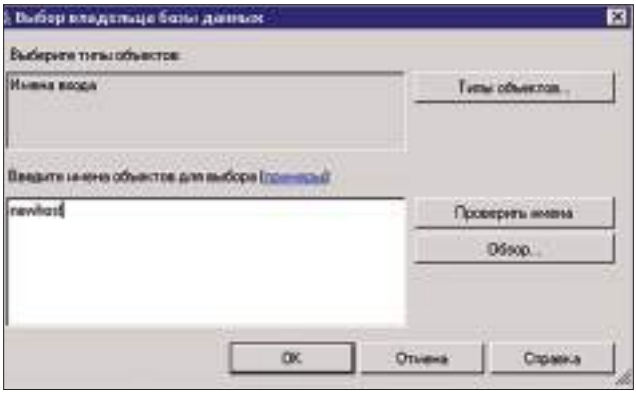

Рис. 47. Имя входа БД

Если такое имя входа найдено, оно отобразится в окне «Создание базы данных» (рис. 48).

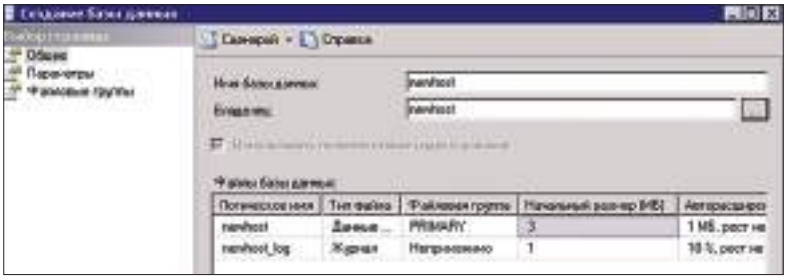

Рис. 48. Поиск имени входа БД

Нажимаем OK для создания базы данных.

Снова открываем папку «Безопасность» → Имена входа (рис. 42), отыскиваем недавно введенное имя входа и дважды щелкаем по нему мышью. Открывается уже знакомое нам окно. В окне «База данных по умолчанию» воспользуемся выпадаю-

щим списком (рис. 49) и вместо master по умолчанию выберем недавно созданную БД.

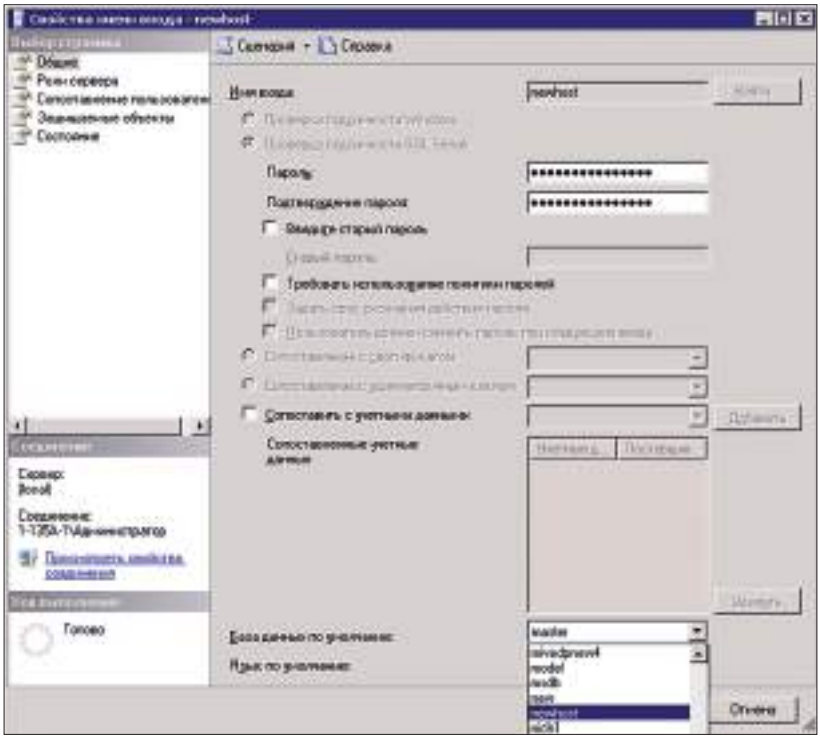

Рис. 49. Замена признака master

По завершении нажимаем OK для закрытия окна.

3.6. Создание серверных параметров ИС

Рассмотрим создание непосредственно ИС-типа — сайта средствами сервера IIS (Internet Information Services) и настройку параметров запуска сайта. В IIS входят следующие серверы. Web-сервер является основным и наиболее часто используемым компонентом IIS. FTP-сервер — сервер для обмена файлами. SMTP-сервер — сервер отправки почты (устанавливается как Feature, но управляется консолью IIS6).

Основным компонентом IIS является веб-сервер, который позволяет размещать в Интернете сайты. IIS поддерживает протоколы HTTP, HTTPS, FTP, POP3, SMTP, NNTP.

Возвращаемся в окно «Диспетчер сервера», щелкнув по кнопке «Диспетчер сервера» на панели задач «Инструментарий администрирования».

Открываем куст «Диспетчер сервера» → Роли → Веб-сервер (IIS) и щелкаем по пиктограмме «Веб-сервер» (IIS). В правой части окна появится рабочее окно оснастки «Диспетчер служб IIS». Раскрываем куст 1-135А-1 (рис. 50), затем раскрываем куст «Сайты».

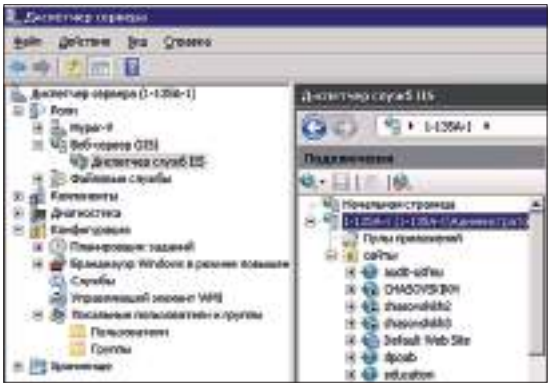

Рис. 50. Диспетчер службы IIS

Дерево «Сайты» содержит список всех сайтов, функционирующих на данном сервере.

Для создания нового сайта необходимо щелкнуть правой кнопкой мыши по папке «Сайты» и выбрать пункт «Добавить веб-сайт» (рис. 51).

Рис. 51. Добавление веб-сайта

В открывшемся окне вводим имя сайта, а также физический путь. Физический путь должен совпадать с адресом каталога, созданного ранее (рис. 33). И, соответственно, совпадать с именем пользователя, созданного нами ранее (рис. 34).

В нашем случае путь — это C:\inetpub\wwwroot\sites\LocalUser\newhost (рис. 52).

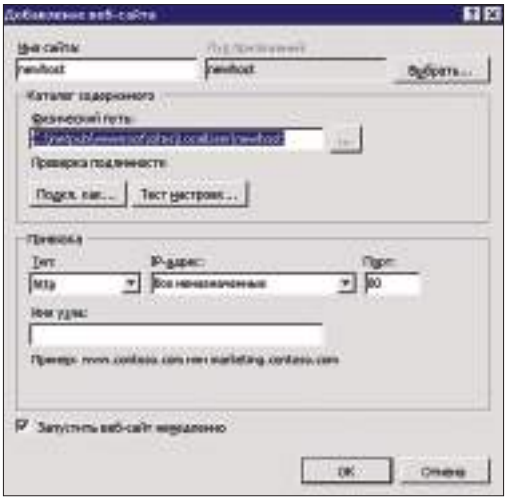

Рис. 52. Физический путь к папке сайта

В группе элементов управления «Привязка» необходимо ввести параметры, обусловливающие разрешение входящего http-запроса к конкретному сайту.

В поле «Тип» оставляем протокол http, в поле «IP-адрес» выбираем выделенный адрес, например: 79.110.248.214.

Стандартным портом для http-протокола считается порт 80, он функционирует на всех устройствах, имеющих задачу работы с web-сайтами.

В современной практике интернета имеет место проблема, что количество сайтов превышает количество IPv4-адресов. А как следствие, современное программное обеспечение браузера и сервера позволяет по одному IP-адресу размещать несколько сайтов с разными доменными именами. Сервер IIS выделяет из заголовка запроса имя сайта и выдает браузеру нужный. Та-

ким образом, можно запустить в IIS теоретически неограниченное количество сайтов, введя IP-адрес 79.110.248.214 и порт 80, но разные имена узла, вроде (имена виртуальные):

site1.usue.ru
site2.usue.ru
site3.usue.ru

Под доменами третьего уровня можно задавать символьно-логичные имена, они считываются поисковиками и открываются на любом компьютере. Но! Для каждого такого сайта нужно дополнительно в файле на DNS-сервере написать строчку вида: site1.usue.ru. 3600 IN CNAME usue.ru.

Практически можно разнести сайты по разным портам и надеяться, что эти порты не будут заняты каким-либо приложением. Для этого в поле «Имя узла» вводим «usue.ru», а в поле «Порт» стираем 80 и вводим какой-нибудь случайный и надеемся, что он свободен. Например, 8105 (рис. 53).

Рис. 53. Добавление сайта

Если свободен, то нам повезло, и к сайту можно будет обратиться по адресу: **usue.ru:8105.** Если нет, то нельзя, пробуем другой порт.

Для завершения настройки необходимо щелкнуть по пиктограмме «Пулы приложений» в диспетчере служб IIS и в открывшемся списке найти позицию с именем, соответствующим имени недавно созданного сайта (рис. 54).

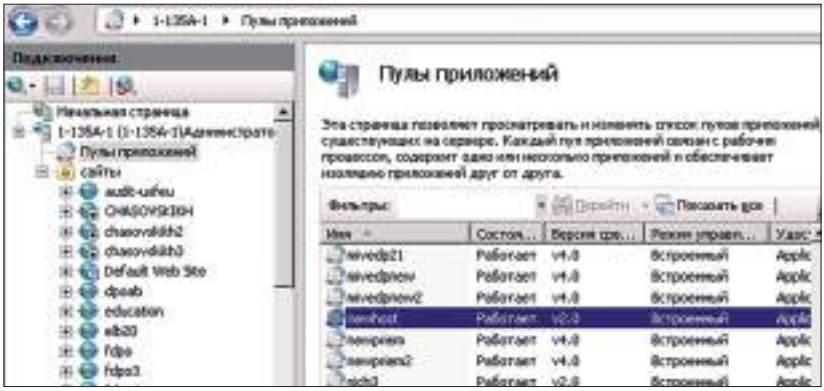

Рис. 54. Завершение настройки сайта на сервере

По умолчанию IIS создает пул приложений с какой-то версией. Если это не v4.0, то необходимо изменить (рис. 55).

Для изменения дважды щелкаем по имени пула левой кнопкой мыши, в появившемся окне выбираем v4.0 и нажимаем OK.

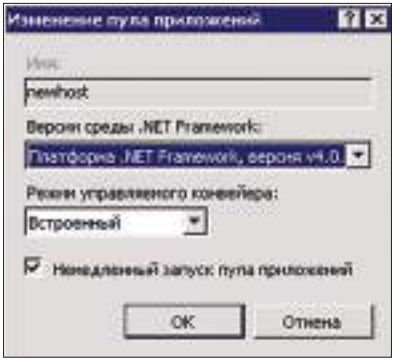

Рис. 55. Версия пула приложений

В случае необходимости перезапускаем пул приложений и сайт кнопкой «Перезапуск» в правой колонке окна диспетчера (рис. 56).

Рис. 56. Перезапуск пула приложений и сайта

Выводы к главе 3

Для ИС типа «Сайт» рассмотрены функции администрирования для выделения хостинга на сервере, к которому подключена ИС.

Определено, что хостинг — это пространство на сервере, где будут храниться данные и файлы для функционирования ИС типа «Сайт».

Прежде всего, хостинг необходим всем, вне зависимости от того, создаете вы блог, интернет-магазин, крупный бизнес-проект.

Основное преимущество выделенного хостинга заключается в том, что пользователь получает собственный сервер для своего сайта. Его можно настраивать так, как захочется, выбирать ПО и операционную систему в соответствии со своими потребностями.

Рассмотрены вопросы безопасности хостинга, так как любая ИС типа «Сайт» нуждается хотя бы в минимальном уровне защиты средствами операционной системы сервера.

Для некоторых видов сайтов необходима усиленная защита, например, на них могут обрабатываться данные банковских карт или храниться персональные данные пользователей. Для таких хостингов должен применяться пакет программ антивируса Касперского для серверов.

Вопросы для самоконтроля

1. Хостинг БД и программ сайта располагаются в одном пространстве сервера или ...?
2. Как подключиться к серверу?
3. Какой инструментарий сервера необходим для выделения хостингов?
4. Создание каталога на диске сервера с файловым содержимым сайта.
5. Создание на сервере имени пользователя для доступа по ftp.
6. Имя пользователя и имя каталога. Определение защиты доступа к хостингу.
7. Создание пользователя базы данных Microsoft SQL.
8. Создание базы данных и ее защита паролем доступа.
9. Создание серверных параметров ИС.

ГЛАВА 4

Windows Server 2019 и инструменты администрирования

Центральным элементом ИС является SQL Server серверной операционной системы. В учебном пособии мы будем рассматривать Windows Server 2019 (WS-2019) с SQL Server 2019 (SQL-2019). Главным в выполнении задач администрирования ИС является правильная настройка (SQL-2019).

Полная документация (SQL-2019), включая описание всех функций администрирования, размещена по ссылке: https://docs.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver15.

4.1. Современная платформа для разработки ИС

Разработка ИС в рамках цифровой экономики и прежде всего с условиями сквозных технологий определяет необходимость платформы баз данных для решения современных задач. В составе WS-2019 такая платформа представлена SQL-2019.

Впервые в серверной операционной системе ИС предоставлены следующие возможности:

— платформа баз данных поддерживает новые типы приложений, интегрированных с машинным обучением, — приложений, которые масштабируемы, безопасны и интегрированы с базой данных. Службы машинного обучения SQL-2019 (SQL Server Machine Learning Services) включают эти новые возможности SQL-2019;

— поддерживается широкий спектр языков и драйверов для различных платформ операционных систем, таких как Windows, macOS и Linux, а также обеспечение совместимости между ними;

— определены графовые базы данных, интегрированные с SQL-2019, которые позволяют при разработке ИС реализо-

вывать различные модели данных, такие как социальные сети, и выполнять запросы к ним, используя знакомый язык T-SQL (Transact-SQL);

— SQL-2019 поддерживает кодировку UTF-8 (кодировка символов переменной ширины, используемая для электронной связи и определяемая стандартом Unicode) благодаря новым правилам сравнения и хранения данных;

— язык T-SQL позволяет устанавливать и использовать новые языки, такие как Java, интегрированные с данными SQL-2019.

4.2. Графовая база данных

Концепция реляционной базы данных охватывает все типы моделей проектирования, схемы данных и приложения. Тем не менее существуют определенные разновидности моделей данных, разработанные для решения задач в определенной области, которые не всегда полностью соответствуют стандартной реляционной модели и языку SQL.

Такие модели обычно включают иерархические, сетевые или сложные отношения данных «многие ко многим».

Некоторые разработчики ИС все еще пытались использовать реляционную базу данных, чтобы встроить ее в модель графа и писать сложные запросы T-SQL для навигации по графу.

Были даже созданы специальные проекты для графовых данных, такие как популярная графовая база данных с открытым исходным кодом Neo4j (https://github.com/neo4j/neo4j).

Другие платформы баз данных разрабатывали «надстройки» над своими реляционными базами данных, чтобы обеспечить возможность работы с данными графов.

Одним из огромных преимуществ графовой базы данных в SQL-2019 является то, что она обладает всеми мощными возможностями SQL-2019.

К этим возможностям относятся HADR (High Availability Disaster Recovery), безопасность, производительность и все функции ядра.

HADR — инструмент БД DB2, реализующий зеркалирование баз данных, позволяющий в случае падения базы автоматически переключать пользователей ИС на резервный сервер.

Другие платформы используют иной подход к решению данной проблемы. Вместо того чтобы включать эти возможности в ядро базы данных, они рассматривают такие функции как надстройки или полностью отдельные продукты.

Графовая база данных в SQL-2019 использует таблицы для представления узлов и ребер в графовой модели с использованием расширений T-SQL.

Использование термина «база данных» здесь оправдано, поскольку графовая база данных не является отдельной базой данных в SQL-2019.

В графовой базе данных узел — это сущность или объект, а ребро — это отношение между узлами.

Таким образом, графовая база данных представляет собой набор таблиц узлов и ребер, а также данных и метаданных, которые связывают их вместе.

SQL-2019 поддерживает расширения языка T-SQL для определения таблицы узлов или ребер с помощью синтаксиса AS NODE или AS EDGE для оператора CREATE TABLE. Полное описание синтаксиса команды, используемой для создания таблицы узлов или ребер, можно найти по ссылке: https://docs.microsoft.com/en-us/sql/t-sqL/statements/create-tabLe-sqL-graph.

Кроме того, SQL-2019 поддерживает новое ключевое слово T-SQL — MATCH, позволяющее перемещаться по таблицам узлов и ребер, как часть оператора SELECT.

Многие из вас, вероятно, являются новичками в использовании графовой базы данных в SQL-2019, поэтому рассмотрим простой пример, чтобы продемонстрировать всю мощь этой возможности.

Рассмотрим социальную сеть.

Сеть по своей природе представляет собой соединение объектов, обычно моделируемое в виде графа. Рассмотрим сеть друзей, изображенную на рис. 57.

В этой модели графа узлами являются Человек, Город и Ресторан. Стрелки представляют отношения между узлами; это ребра графа. Обратите внимание на особые отношения под названием «Друзья», которые связывают людей друг с другом. Построить эту модель, используя набор реляционных таблиц, не так уж и сложно, но обход графа с использованием традиционных запросов T-SQL становится достаточно трудоемкой задачей.

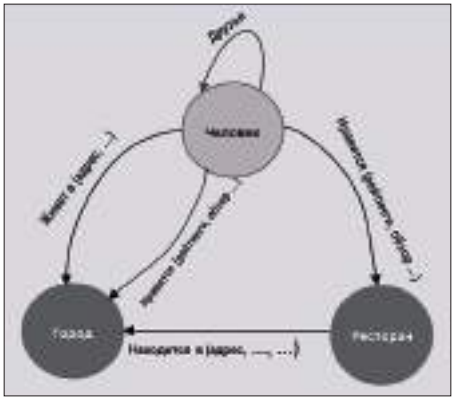

Рис. 57. Сеть друзей

Давайте создадим пример графовой базы данных с использованием предыдущей модели, чтобы вы могли познакомиться с основами, а затем посмотреть, что нового в SQL-2019.

Для нашего примера допустим, что некто Джон (John) является пользователем социальной сети. Он дружит с Мэри (Mary) и Джули (Julie), но еще не знает, с кем они дружат (и с кем дружат их друзья и т. д.). Он хочет расширить свою социальную сеть, а также узнать, какие рестораны нравятся его друзьям.

Рассмотрим «социальную сеть друзей», изображенную на рис. 58.

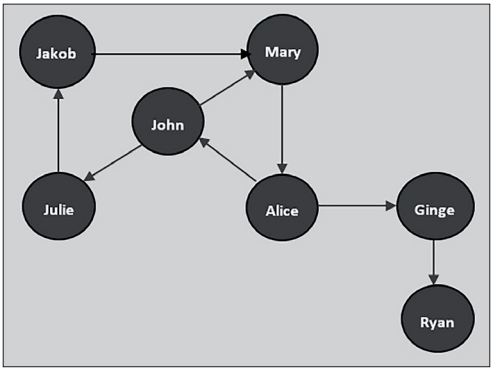

Рис. 58. Социальная сеть друзей

Джон не знает всей социальной сети, поэтому ему нужно использовать графовую базу данных для навигации.

Все сценарии для примеров использования графовой базы покажем на следующем примере.

Пример базы данных SQL Graph — социальная сеть

В этом примере покажем, как построить базу данных графа в виде узла и ребер для имитации социальной сети.

Шаги 1–7 можно выполнить с помощью SQL Server 2017.

Для шагов 8–9 требуется SQL Server 2019.

Шаг 1. Создаем графические таблицы для социальной сети.

Социальная сеть будет состоять из следующих узловых таблиц:

— Person;

— Restaurant;

— City.

И следующие таблицы EDGE-браузера:

— friendOf — определяет отношения между людьми;

— liveIn — в каком городе живут люди;

— locateIn — в каком городе находится ресторан;

— likes — какие рестораны нравятся людям и как они их оценивают.

In [1]:

```
USE master;
GO
DROP DATABASE IF EXISTS socialnetwork;
GO
CREATE DATABASE socialnetwork;
GO
USE socialnetwork;
GO
DROP TABLE IF EXISTS Person;
DROP TABLE IF EXISTS Restaurant;
DROP TABLE IF EXISTS City;
DROP TABLE IF EXISTS likes;
DROP TABLE IF EXISTS friendOf;
```

```
DROP TABLE IF EXISTS livesIn;
DROP TABLE IF EXISTS locatedIn;
GO
CREATE TABLE Person (id INTEGER PRIMARY KEY NOT NULL, name
VARCHAR(100) NOT NULL) AS NODE;
CREATE TABLE Restaurant (id INTEGER PRIMARY KEY NOT NULL,
name VARCHAR(100) NOT NULL, city VARCHAR(100) NOT NULL) AS
NODE;
CREATE TABLE City (id INTEGER PRIMARY KEY NOT NULL, name
VARCHAR(100) NOT NULL, stateName VARCHAR(100) NOT NULL) AS
NODE;
CREATE TABLE friendOf AS EDGE;
CREATE TABLE livesIn AS EDGE;
CREATE TABLE locatedIn AS EDGE;
CREATE TABLE likes (rating INTEGER) AS EDGE;
GO
```

Шаг 2. Заполняем социальную сеть таблицами узлов.

Введем данные в таблицы узлов. Это люди, рестораны и города.

In [2]:

```
INSERT INTO Person VALUES (1,'John');
INSERT INTO Person VALUES (2,'Mary');
INSERT INTO Person VALUES (3,'Alice');
INSERT INTO Person VALUES (4,'Jacob');
INSERT INTO Person VALUES (5,'Julie');
INSERT INTO Person VALUES (6, 'Ginger');
INSERT INTO Person VALUES (7, 'Ryan');
GO
INSERT INTO Restaurant VALUES (1,'Taco Dell','Bellevue');
INSERT INTO Restaurant VALUES (2,'Ginger and
Spice','Seattle');
INSERT INTO Restaurant VALUES (3,'Noodle Land', 'Redmond');
INSERT INTO RestaUrant VALUES (4,'BBQ Heaven', 'North
Richland Hills');
GO
INSERT INTO City VALUES (1,'Bellevue','wa');
```

```
INSERT INTO City VALUES (2,'Seattle','wa');
INSERT INTO City VALUES (3,'Redmond','wa');
INSERT INTO City VALUES (4, 'North Richland Hills', 'tx');
GO
```

Шаг 3. Заполняем отношения в социальной сети через EDGE-таблицы.

Введем данные в EDGE-таблицы. Обратите внимание на использование ключевого слова $node_id для вставки значения из таблицы узлов.

In [3]:

```
INSERT INTO friendOf VALUES ((SELECT $NODE_ID FROM Person
WHERE ID = 1), (SELECT $NODE_ID FROM Person WHERE ID = 2));
INSERT INTO friendOf VALUES ((SELECT $NODE_ID FROM Person
WHERE ID = 2), (SELECT $NODE_ID FROM Person WHERE ID = 3));
INSERT INTO friendOf VALUES ((SELECT $NODE_ID FROM Person
WHERE ID = 3), (SELECT $NODE_ID FROM Person WHERE ID = 1));
INSERT INTO friendOf VALUES ((SELECT $NODE_ID FROM Person
WHERE ID = 4), (SELECT $NODE_ID FROM Person WHERE ID = 2));
INSERT INTO friendOf VALUES ((SELECT $NODE_ID FROM Person
WHERE ID = 5), (SELECT $NODE_ID FROM Person WHERE ID = 4));
INSERT INTO friendOf VALUES ((SELECT $NODE_ID FROM Person
WHERE ID = 3), (SELECT $NODE_ID FROM Person WHERE ID = 6));
INSERT INTO friendOf VALUES ((SELECT $NODE_ID FROM Person
WHERE ID = 1), (SELECT $NODE_ID FROM Person WHERE ID = 5));
INSERT INTO friendOf VALUES ((SELECT $NODE_ID FROM Person
WHERE ID = 6), (SELECT $NODE_ID FROM Person WHERE ID = 7));
GO

INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person
WHERE ID = 1),
      (SELECT $node_id FROM City WHERE ID = 1));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person
WHERE ID = 2),
      (SELECT $node_id FROM City WHERE ID = 2));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person
WHERE ID = 3),
```

```
      (SELECT $node_id FROM City WHERE ID = 3));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person
WHERE ID = 4),
      (SELECT $node_id FROM City WHERE ID = 3));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person
WHERE ID = 5),
      (SELECT $node_id FROM City WHERE ID = 1));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person
WHERE ID = 6),
      (SELECT $node_id FROM City WHERE ID = 4));
INSERT INTO livesIn VALUES ((SELECT $node_id FROM Person
WHERE ID = 7),
      (SELECT $node_id FROM City WHERE ID = 4));
GO

INSERT INTO locatedIn VALUES ((SELECT $node_id FROM
Restaurant WHERE ID = 1),
      (SELECT $node_id FROM City WHERE ID =1));
INSERT INTO locatedIn VALUES ((SELECT $node_id FROM
Restaurant WHERE ID = 2),
      (SELECT $node_id FROM City WHERE ID =2));
INSERT INTO locatedIn VALUES ((SELECT $node_id FROM
Restaurant WHERE ID = 3),
      (SELECT $node_id FROM City WHERE ID =3));
INSERT INTO locatedIn VALUES ((SELECT $node_id FROM
Restaurant WHERE ID = 4),
      (SELECT $node_id FROM City WHERE ID =4));
GO

INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE
ID = 1),
      (SELECT $node_id FROM Restaurant WHERE ID = 1),9);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE
ID = 2),
      (SELECT $node_id FROM Restaurant WHERE ID = 2),9);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE
ID = 3),
      (SELECT $node_id FROM Restaurant WHERE ID = 3),9);
```

```
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE
ID = 4),
      (SELECT $node_id FROM Restaurant WHERE ID = 3),6);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE
ID = 5),
      (SELECT $node_id FROM Restaurant WHERE ID = 3),9);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE
ID = 6),
      (SELECT $node_id FROM Restaurant WHERE ID = 4),10);
INSERT INTO likes VALUES ((SELECT $node_id FROM Person WHERE
ID = 7),
      (SELECT $node_id FROM Restaurant WHERE ID = 4),10);
GO
```

Шаг 4. Кто мои друзья?

Джон хочет найти друзей в своей сети. Как он находит своих ближайших друзей?

In [4]:

```
USE socialnetwork
GO
SELECT Person2.name AS FriendName
FROM Person Person1, friendOf, Person Person2
WHERE MATCH(Person1-(friendOf)→Person2)
AND Person1.name = 'John';
GO
```

FriendName
Mary;
Julie;

Шаг 5. Кто друзья моих друзей?

John хочет знать, кто является непосредственными друзьями Mary и Julie. Обратите внимание, что здесь используется обход графа друзей на втором уровне. Теперь John знает, что он мог бы добавить к своей сети друзей Alice и Jacob, используя Mary и Julie ленты с постами в соцсетях.

In [5]:

```
SELECT person1.name +' is friends with ' + person2.name, + 'who
is friends with '+ person3.name
FROM Person person1, friendOf friend1, Person person2,
friendOf friend2, Person person3
WHERE MATCH(person1-(friend1)person2-(friend2) → person3)
AND person1.name = 'John';
GO
```

John дружит с Mary, которая дружит с Alice.
John дружит с Julie, которая дружит Jacob.

Шаг 6. Найдите рестораны, которые нравятся моим друзьям.
John ищет новый ресторан и хочет знать, какие рестораны нравятся его друзьям.

In [6]:

```
SELECT person2.name, Restaurant.name, likes.rating
FROM Person person1, Person person2, likes, friendOf,
Restaurant
WHERE MATCH(person1-(friendOf) → person2-(likes) → Restaurant)
AND person1.name='John';
GO
```

Имя	Имя	Рейтинг
Mary	Ginger и Spice	9
Julie	Noodle Land	9

Шаг 7. Кому нравятся рестораны там, где они живут?
John хочет узнать, кто любит рестораны в городах, в которых они живут, чтобы узнать, какие еще есть варианты.

In [7]:

```
SELECT Person.name, Restaurant.name, likes.rating, City.name
FROM Person, likes, Restaurant, livesIn, City, locatedIn
WHERE MATCH (Person-(likes) → Restaurant-(locatedIn) → City AND
Person-(livesIn) → City);
GO
```

Имя	Имя	Рейтинг	Имя
John	Taco Dell	9	Bellevue
Mary	Ginger and Spice	9	Seattle
Alice	Noodle Land	9	Redmond
Jacob	Noodle Land	6	Redmond
Ginger	BBQ Heaven	10	North Richland Hills
Ryan	BBQ Heaven	10	North Richland Hills

Шаг 8. Узнайте мою возможную социальную сеть.

John хочет знать, какие еще есть возможные друзья, чтобы расширить свою социальную сеть. Он хочет глубже погрузиться в уровни графа сети, не переходя по одному уровню за раз. SHORTEST_PATH() может помочь.

Теперь John видит, что Alice дружит с ним, но он не указал Alice в списке друзей. Он также замечает, что Ginger (и косвенно Ryan) дружит с Alice. Он заметил, что Ginger и Ryan из Техаса, и у него скоро командировка туда. Возможно, он мог бы поговорить с Alice о том, какие Ginger и Ryan и стоит ли ему попытаться подружиться с ними (и, возможно, попробовать их любимое место для барбекю).

[8]:

```
SELECT
       Person1.name AS PersonName,
       STRING_AGG(Person2.name, '→') WITHIN GROUP (GRAPH
       PATH) AS Friends
FROM
       Person AS Person1,
       friendOf FOR PATH AS fo,
       Person FOR PATH AS Person2
WHERE MATCH(SHORTEST_PATH(Person1(-(fo)→Person2)+))
AND Person1.name = 'John';
GO
```

Имя	Друг
John	Mary
John	Julie
John	Mary → Alice

John	Julie → Jacob
John	Mary → Alice → John
John	Mary → Alice → Ginger
John	Mary → Alice → Ginger → Ryan

Шаг 9. Найдите самый быстрый способ подружиться.
SHORTEST_PATH() поможет найти самый быстрый способ подружиться.

```
                                                                    In [9]:
SELECT PersonName, Friends
FROM (
SELECT
       Person1.name AS PersonName,
       STRING_AGG(Person2.name, ' → ') WITHIN GROUP (GRAPH
       PATH) AS Friends,
       LAST_VALUE(Person2.name) WITHIN GROUP (GRAPH PATH) AS
       LastNode
FROM
       Person AS Person1,
       friendOf FOR PATH AS fo,
       Person FOR PATH AS Person2
WHERE MATCH(SHORTEST_PATH(Person1(-(fo)→Person2)+))
AND Person1.name = 'Jacob'
) AS Q
WHERE Q.LastNode = 'Alice';
GO
```

Имя	Друг
Jacob	Mary → Alice

Также можно использовать сценарий T-SQL socialnetwork.sql для выполнения шагов построения графа с применением SSMS или ADS.

Расширение возможностей работы с графовыми данными в SQL-2019 включает несколько улучшений, которые помогли сделать графовую базу данных более мощной и привлекательной

платформой для работы с данными графов по сравнению с другими продуктами. Эти возможности включают в себя:

— обход пути графа с использованием нового синтаксиса SHORTEST-PATH();

— поддержку производных таблиц (derived tables) и представлений (views) в графовой базе данных;

— ограничения ребер, необходимые для построения правильных отношений внутри графа;

— использование оператора T-SQL MERGE для поддержки таких сценариев, как upsert.

SHORTEST_PATH()

Одной из наиболее распространенных задач, которые необходимо решить с помощью графовой базы данных, является рекурсивный просмотр данных графа без необходимости вручную переходить на каждый уровень. SQL-2017 не поддерживает эту возможность, но SQL-2019 обеспечивает ее поддержку посредством нового синтаксиса T-SQL SHORTEST_PATH().

Ограничения ребер

Хотя синтаксис NODE и EDGE T-SQL обеспечивает отличный новый способ построения данных графа с использованием таблиц SQL Server, в SQL-2017 не существовало способа обеспечить целостность данных узлов и ребер. Аналогично тому, как концепция ограничений внешнего ключа в таблицах SQL-2019 поддерживает целостность данных в таблицах, SQL-2019 поддерживает возможность обеспечения целостности данных для узлов и ребер.

Исследуйте только что созданную социальную сеть, используя примеры из этого раздела. Было бы неплохо убедиться, что любые данные для таблицы ребер friendOf должны быть взяты из соответствующей строки в таблице Person. Ограничения ребер предоставляют такую возможность.

Кроме того, ограничения ребер поддерживают правильные связи внутри сети. В нашей модели социальной сети человеку может нравиться ресторан, но ресторану не может нравиться человек. Ограничение ребер также может обеспечить выполнение этого правила. Кроме того, ограничения ребер гарантируют,

что ребра не останутся «оборванными», т. е. ссылающимися на несуществующий объект. Ограничения ребер не позволяют удалить узел, который является частью отношения ребер, если такие данные присутствуют в таблице ребер. Опять же, подобное поведение аналогично ограничениям внешних ключей в традиционных реляционных таблицах.

Использование MERCE с таблицами графов

SQL-2019 использует оператор T-SOL под названием MERGE, который выполняет операции вставки, обновления или удаления целевой таблицы на основе результатов объединения с исходной таблицей. В SQL-2017 вы можете использовать инструкцию MERGE для консолидации операций DML в таблицах узлов, но не в таблицах ребер. SQL-2019 теперь предоставляет возможность применять MERGE и для таблиц ребер.

4.3. Службы машинного обучения SQL Server

Платформа машинного обучения в SQL получила название «SQL Server 2016 R Services».

В SQL Server 2017 была представлена интеграция с языком программирования Python, в основу которой были положены те же концепции и архитектура. Благодаря этим новым возможностям R и Python совместно стали службами машинного обучения SQL Server (SQL Server Machine Learning Services).

Хотя изменения в службах машинного обучения SQL Server (ML Services) для SQL Server 2019 не являются кардинальными, эти возможности являются новыми по следующим причинам:

— понимание того, как работают ML Services, и преимущества их использования позволяют принять решение о том, подходят ли эти возможности для приложения;

— можно больше доверять ML Services для SQL Server, поскольку существенно расширились функции интеграции, безопасности и управлении службами;

— архитектура, называемая Extensibility Framework, используемая для ML Services, та же, что и для так называемых языковых расширений SQL Server, новых для SQL Server 2019.

Службы SQL Server ML Services предлагают новую возможность для создания и использования масштабируемых приложений машинного обучения в соответствии со следующими концепциями:

— приложения машинного обучения выполняются на том же компьютере, что и SQL Server, но в независимых от SQLSERVR.EXE процессах;

— SQL Server предоставляет интерфейс T-SQL посредством системной хранимой процедуры sp_execute_external_script для выполнения кода машинного обучения;

— SQL Server предоставляет архитектуру, делающую возможными интеллектуальный обмен данными и масштабируемость, используя специальный программный код для машинного обучения.

Рисунок 59 иллюстрирует архитектуру служб SQL Server ML Services, называемую расширяемой архитектурой служб машинного обучения SQL Server (Extensibility architecture in SQL Server Machine Learning Services).

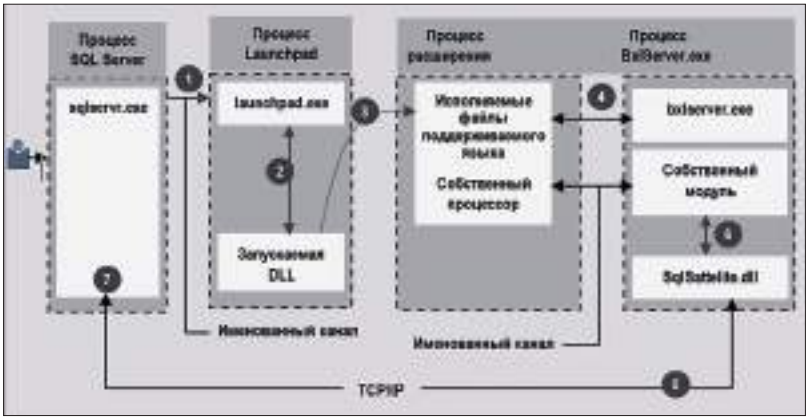

Рис. 59. Архитектура служб SQL Server ML Services

Для лучшего понимания архитектуры служб SQL Server ML Services расширим визуализируемые функции, как показано на рис. 60.

Функции расширенной службы SQL Server ML Services следующие.

1. Пользователь выполняет сценарий sp_execute_external_script, выбрав язык программирования (R или Python), сценарий и другие параметры, такие как запрос T-SQL, для выполнения сценария. SQL Server вызывает отдельную программу с названием Launchpad (служба в Windows или демон в Linux) через именованные каналы, передавая все соответствующие данные (например, сценарий R или Python).

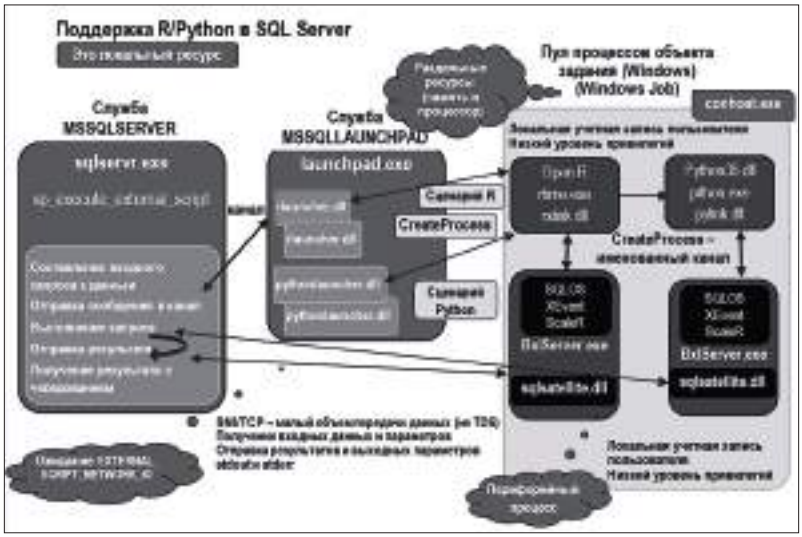

Рис. 60. Функционально расширенная служба SQL Server ML Services

2. Launchpad содержит код для выполнения DLL, соответствующий языку R или Python. Launchpad использует модель рабочего потока, аналогичную ядру SQL Server. Фактически он загружает систему SQLOS, которую SQL Server использует для служб ОС.

3. DLL-библиотека Launchpad запустит или создаст новый процесс для соответствующего языка (rterm.exe для R или python.exe для Python).

4. Другой процесс выполняет создание дочернего потока, который называется bxlserver.exe (часто его называют перифе-

рийным процессом). Этот программный модуль будет взаимодействовать с rterm.exe или python.exe для обмена данными.

5. Файл bxlserver.exe обменивается данными с ядром SQL Server, используя частный канал TCP (это не аналогично подключению клиента к SQL Server) для получения данных из запроса T-SQL, выполняемого для передачи данных в программу на R или Python. Этот обмен данными выполняется с чередованием. Это означает, что механизм SQL Server может получить строки для запроса T-SQL, чтобы передать их в программный модуль машинного обучения, и в то же время получить результаты обратно. DLL, которая поддерживает такой обмен, называется sqlsatellite.dll.

6. Файл sqlsatellite.dll работает с модулем в bxlserver.exe для обмена данными с rterm.exe или python.exe.

7. Все результаты (включая сообщения stdout) из программного модуля rterm.exe или python.exe передаются обратно на SQL Server через канал TCP.

В результате этого пользователь выполняет sp_execute_external_script и получает обратно результаты в виде таблицы (например, набора результатов SELECT) вместе с сообщениями stdout. У этой процедуры есть также специальные параметры оператора для выходных параметров и многое другое.

Ключевая концепция этого улучшения заключается в том, что код R или Python выполняется на том же компьютере, что и SQL Server (близко к данным), и SQL Server может эффективно обмениваться данными с программным кодом (сетевой трафик для обмена данными не используется).

Поясним на примере взаимодействие запроса T-SQL (или входной запрос) и программы на R или Python.

Создадим прогностическую модель с помощью Python и SQL Server ML Services.

Последовательно выполним следующие действия:

— настроим среду;

— создадим скрипт ML с помощью Python;

— развернем сценарий машинного обучения с помощью SQL Server.

Прогнозное моделирование — это мощный способ добавить интеллектуальные возможности к приложению. Это позволяет приложениям прогнозировать результаты на основе новых дан-

ных. Процесс включения прогнозной аналитики в приложения включает в себя два основных этапа: **обучение модели** и **развертывание модели.**

Шаг 1. Устанавливаем SQL Server со службами машинного обучения в базе данных.

Загружаем исполняемый файл SQL Server 2016 (эта версия поддерживает только R для машинного обучения).

1. Запускаем исполняемый файл SQL Server 2016, активируя установщик SQL.
2. Нажимаем «*Принять*» после прочтения условий лицензии.
3. На странице «*Выбор функций*» выбираем: «Службы R» (в базе данных) для SQL Server 2016 или «Службы машинного обучения» (в базе данных) для SQL Server 2017.
4. Выбираем R/Python или оба.
5. Если вы выбрали R: на странице «*Согласие на установку Microsoft R Open*» нажмите «*Принять*».
6. Если вы выбрали Python: на странице «*Согласие с Python*» нажмите «*Принять*».
7. Нажимаем «Установить», чтобы продолжить установку.

Если не было ошибок, то установлен SQL Server со службами машинного обучения в базе данных, которые работают локально на вашем компьютере с Windows.

Шаг 2. Устанавливаем SQL Server Management Studio (SSMS).

Загружаем и устанавливаем студию управления SQL Server — SSMS. SSMS будем использовать как инструмент для простого управления объектами базы данных и сценариями.

Шаг 3. Включаем выполнение внешнего скрипта.

Запускаем SSMS и открываем новое окно запроса. Затем выполняем приведенный ниже сценарий, чтобы ваш экземпляр мог запускать сценарии Python в SQL Server.

SQL

```
EXEC sp_configure 'external scripts enabled', 1;
RECONFIGURE WITH OVERRIDE
```

Копировать

Перезапустить в SSMS, щелкнув правой кнопкой мыши имя экземпляра в обозревателе объектов и выбрав *Restart*.

Теперь мы включили выполнение внешнего скрипта, чтобы мы могли запускать код Python внутри SQL Server.

Шаг 4. Устанавливаем и настраиваем среду разработки Python.

Необходимо установить Python IDE.

Шаг 5. Устанавливаем удаленные клиентские библиотеки Python.

В примере рассмотрим бизнес по аренде лыж, и мы хотим предсказать количество прокатов, которое у нас будет в будущем. Эта информация поможет нам подготовиться с точки зрения запасов, персонала и оборудования.

Во время **обучения модели** вы создаете и обучаете прогностическую модель, показывая ей образцы данных вместе с результатами. Затем вы сохраняете эту модель, чтобы использовать ее позже, когда вы хотите делать прогнозы на основе новых данных.

После установки SQL Server со службами ML и настройки Python IDE на вашем компьютере можно приступить к обучению прогностической модели с помощью Python.

Шаг 1. Загрузите образцы данных.

Восстанавливаем образец БД. Набор данных, используемый в этом примере, размещен в таблице SQL Server. Таблица содержит данные об аренде за предыдущие годы.

1. Загружаем файл резервной копии (.bak) здесь и сохраняем его в расположении, доступном для SQL Server. Например, в папке, где установлен SQL Server. Пример пути: C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Backup.

2. После сохранения файла откройте SSMS и новое окно запроса, чтобы выполнить следующие команды для восстановления БД. Обязательно измените пути к файлам и имя сервера в сценарии.

SQL

```
USE master;
```

```
GO
RESTORE DATABASE TutorialDB
       FROM DISK = 'C:\Program Files\Microsoft SQL Server\
       MSSQL13.MSSQLSERVER\MSSQL\Backup\TutorialDB.bak'
       WITH
       MOVE 'TutorialDB' TO 'C:\Program Files\Microsoft SQL
       Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\TutorialDB.mdf'
      ,MOVE 'TutorialDB_log' TO 'C:\Program Files\Microsoft
       SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\TutorialDB.
       ldf';
GO
```

Таблица с именем rent_data, содержащая набор данных, должна существовать в восстановленной базе данных SQL Server. Вы можете убедиться в этом, запросив таблицу в SSMS.

SQL

```
USE tutorialdb;
SELECT * FROM [dbo].[rental_data];
```

Теперь у вас есть база данных и данные для обучения модели.

Шаг 2. Исследуйте данные с помощью Python.

Загружать данные из SQL Server в Python очень просто. Итак, давайте попробуем.

Откройте новый скрипт Python в вашей среде IDE и запустите следующий скрипт. Только не забудьте заменить «MYSQLSERVER» на имя вашего экземпляра базы данных.

Python

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

#If you are running SQL Server 2017 RC1 and above:
from revoscalepy import RxComputeContext, RxInSqlServer,
RxSqlServerData
from revoscalepy import rx_import
```

```
#Connection string to connect to SQL Server named instance
conn_str = 'Driver=SQL Server;Server=MYSQLSERVER;Database=Tuto
rialDB;Trusted_Connection=True;'

#Define the columns we wish to import
column_info = {
      "Year" : { "type" : "integer" },
      "Month" : { "type" : "integer" },
      "Day" : { "type" : "integer" },
      "RentalCount" : { "type" : "integer" },
      "WeekDay" : {
         "type" : "factor",
         "levels" : ["1", "2", "3", "4", "5", "6", "7"]
       },
      "Holiday" : {
         "type" : "factor",
         "levels" : ["1", "0"]
       },
      "Snow" : {
         "type" : "factor",
         "levels" : ["1", "0"]
          }
       }

#Get the data from SQL Server Table
data_source = RxSqlServerData(table="dbo.rental_data",
             connection_string=conn_str,

column_info=column_info)
computeContext = RxInSqlServer(
       connection_string = conn_str,
       num_tasks = 1,
       auto_cleanup = False
)

RxInSqlServer(connection_string=conn_str, num_tasks=1,
auto_cleanup=False)
```

```
# import data source and convert to pandas dataframe
df = pd.DataFrame(rx_import(input_data = data_source))
print("Data frame:", df)
# Get all the columns from the dataframe.
columns = df.columns.tolist()
# Filter the columns to remove ones we don't want to use in
the training
columns = [c for c in columns if c not in ["Year"]]
```

Копировать

Полученные результаты

Rows Processed: 453

Data frame:	Day	Holiday	Month	RentalCount	Snow	WeekDay	Year
0	20	1	1	445	2	2	2014
1	13	2	2	40	2	5	2014
2	10	2	3	456	2	1	2013
3	31	2	3	38	2	2	2014
4	24	2	4	23	2	5	2014
5	11	2	2	42	2	4	2015
6	28	2	4	310	2	1	2013

...

[453 rows × 7 columns]

Теперь вы прочитали данные из SQL Server в Python и изучили их.

Шаг 3. Обучаем модель.

Чтобы предсказать, мы сначала должны найти функцию (модель), которая лучше всего описывает зависимость между переменными в нашем наборе данных. Этот шаг называется **обучением модели.** Набор обучающих данных будет подмножеством всего набора данных.

Мы собираемся создать модель, используя алгоритм линейной регрессии.

Python

```
# Store the variable we'll be predicting on.
target = "RentalCount"
```

```
# Generate the training set. Set random_state to be able to
replicate results.
train = df.sample(frac=0.8, random_state=1)
# Select anything not in the training set and put it in the
testing set.
test = df.loc[~df.index.isin(train.index)]
# Print the shapes of both sets.
print("Training set shape:", train.shape)
print("Testing set shape:", test.shape)
# Initialize the model class.
lin_model = LinearRegression()
# Fit the model to the training data.
lin_model.fit(train[columns], train[target])
```

Копировать

Полученные результаты

```
Training set shape: (362, 7)
Testing set shape: (91, 7)
```

Теперь мы обучили модель линейной регрессии в Python. Давайте использовать его, чтобы предсказать количество арендной платы.

Шаг 4. Прогноз.

Теперь мы собираемся использовать функцию прогнозирования для прогнозирования количества арендной платы, используя наши две модели.

Python

```
# Generate our predictions for the test set.
lin_predictions = lin_model.predict(test[columns])
print("Predictions:", lin_predictions)
# Compute error between our test predictions and the actual
values.
lin_mse = mean_squared_error(lin_predictions, test[target])
print("Computed error:", lin_mse)
```

Копировать

Полученные результаты

```
Predictions:  [ 40. 38.  240. 39.  514. 48.  297. 25.  507. 24.  30.  54.
  40.  26.  30.  34.  42.  390. 336. 37.  22.  35.  55.  350.
  252. 370. 499. 48.  37.  494. 46.  25.  312. 390. 35.  35.
  421. 39.  176. 21.  33.  452. 34.  28.  37.  260. 49.  577.
  312. 24.  24.  390. 34.  64.  26.  32.  33.  358. 348. 25.
  35.  48.  39.  44.  58.  24.  350. 651. 38.  468. 26.  42.
  310. 709. 155. 26.  648. 617. 26.  846. 729. 44.  432. 25.
  39.  28.  325. 46.  36.  50.  63.]
Computed error: 3.59831533436e-26
```

Мы создали модель с помощью Python. Давайте теперь развернем наш код Python, переместив его на SQL Server.

Службы SQL Server ML позволяют обучать и тестировать прогностические модели в контексте SQL Server. Вы создаете программы T-SQL, содержащие встроенные сценарии Python, а механизм базы данных SQL Server позаботится об их выполнении. Поскольку он выполняется в SQL Server, ваши модели можно легко обучить на основе данных, хранящихся в базе данных. Для развертывания мы сохранили свою модель в базе данных и создали хранимую процедуру, которая прогнозирует использование модели.

Теперь переместим только что написанный код Python в SQL Server и развернем нашу прогностическую модель с помощью служб машинного обучения SQL Server. Чтобы развернуть модель, мы сохраняем модель в среде размещения (например, в базе данных) и реализуем функцию прогнозирования, которая использует модель для прогнозирования. Эту функцию можно вызывать из приложений.

Шаг 1. Создаем таблицу для хранения модели.

В SSMS запускаем новое окно запроса и выполняем следующий сценарий T-SQL.

SQL

```
USE TutorialDB;
DROP TABLE IF EXISTS rental_py_models;
GO
CREATE TABLE rental_py_models (
```

```
model_name VARCHAR(30) NOT NULL DEFAULT('default model')
PRIMARY KEY,
model VARBINARY(MAX) NOT NULL
);
GO
```

Копировать

Теперь у вас есть таблица, в которой можно сохранить модель.

Шаг 2. Создаем хранимую процедуру для создания модели. Создаем хранимую процедуру в SQL Server, чтобы использовать код Python, который мы написали в предыдущем модуле, и сгенерировать модель линейной регрессии внутри базы данных. Код Python будет встроен в инструкцию TSQL.

Теперь создаем хранимую процедуру для обучения/генерации модели.

SQL

```
-- Stored procedure that trains and generates a Python model
using the rental_data and a decision tree algorithm
DROP PROCEDURE IF EXISTS generate_rental_py_model;
go
CREATE PROCEDURE generate_rental_py_model (@trained_model
varbinary(max) OUTPUT)
AS
BEGIN
      EXECUTE sp_execute_external_script
      @language = N'Python'
     , @script = N'
from sklearn.linear_model import LinearRegression
import pickle

df = rental_train_data

# Get all the columns from the dataframe.
columns = df.columns.tolist()

# Store the variable well be predicting on.
```

```
target = "RentalCount"

# Initialize the model class.
lin_model = LinearRegression()

# Fit the model to the training data.
lin_model.fit(df[columns], df[target])

#Before saving the model to the DB table, we need to convert
it to a binary object
trained_model = pickle.dumps(lin_model)'

, @input_data_1 = N'select "RentalCount", "Year", "Month",
"Day", "WeekDay", "Snow", "Holiday" from dbo.rental_data where
Year < 2015'
, @input_data_1_name = N'rental_train_data'
, @params = N'@trained_model varbinary(max) OUTPUT'
, @trained_model = @trained_model OUTPUT;
END;
GO

--STEP 3 – Save model to table
TRUNCATE TABLE rental_py_models;

DECLARE @model VARBINARY(MAX);
EXEC generate_rental_py_model @model OUTPUT;

INSERT INTO rental_py_models (model_name, model)
VALUES('linear_model', @model);
```

Копировать

Модель сохраняем в базе данных как бинарный объект.

Шаг 3. Создаем хранимую процедуру для предсказания.

Мы близки к развертыванию нашей модели прогнозирования, чтобы мы могли использовать ее из наших приложений. Этот последний шаг включает в себя создание хранимой про-

цедуры, которая использует нашу модель для прогнозирования количества арендной платы.

Создаем хранимую процедуру, которая прогнозирует, используя нашу модель.

SQL

```
DROP PROCEDURE IF EXISTS py_predict_rentalcount;
GO
CREATE PROCEDURE py_predict_rentalcount (@model varchar(100))
AS
BEGIN
      DECLARE @py_model varbinary(max) = (select model from
      rental_py_models where model_name = @model);

      EXEC sp_execute_external_script
                           @language = N'Python',
                           @script = N'

# Import the scikit-learn function to compute error.
from sklearn.metrics import mean_squared_error
import pickle
import pandas as pd

rental_model = pickle.loads(py_model)

df = rental_score_data

# Get all the columns from the dataframe.
columns = df.columns.tolist()

# variable we will be predicting on.
target = "RentalCount"

# Generate our predictions for the test set.
lin_predictions = rental_model.predict(df[columns])
print(lin_predictions)
```

```
# Compute error between our test predictions and the actual
values.
lin_mse = mean_squared_error(lin_predictions, df[target])
#print(lin_mse)

predictions_df = pd.DataFrame(lin_predictions)

OutputDataSet = pd.concat([predictions_df, df["RentalCount"],
df["Month"], df["Day"], df["WeekDay"], df["Snow"],
df["Holiday"], df["Year"]], axis=1)
'
, @input_data_1 = N'Select "RentalCount", "Year","Month",
"Day", "WeekDay", "Snow", "Holiday" from rental_data where
Year = 2015'
, @input_data_1_name = N'rental_score_data'
, @params = N'@py_model varbinary(max)'
, @py_model = @py_model
with result sets (("RentalCount_Predicted" float,
"RentalCount" float, "Month" float,"Day" float,"WeekDay"
float,"Snow" float,"Holiday" float, "Year" float));

END;
GO
```

Копировать

Создаем таблицу для хранения прогнозов.

SQL

```
DROP TABLE IF EXISTS [dbo].[py_rental_predictions];
GO
--Create a table to store the predictions in
CREATE TABLE [dbo].[py_rental_predictions](
[RentalCount_Predicted] [int] NULL,
[RentalCount_Actual] [int] NULL,
[Month] [int] NULL,
[Day] [int] NULL,
[WeekDay] [int] NULL,
[Snow] [int] NULL,
```

```
[Holiday] [int] NULL,
[Year] [int] NULL
) ON [PRIMARY]
GO
```

Копировать

Выполняем хранимую процедуру для прогнозирования количества аренды.

SQL

```
TRUNCATE TABLE py_rental_predictions;
--Insert the results of the predictions for test set into
a table
INSERT INTO py_rental_predictions
EXEC py_predict_rentalcount 'linear_model';

-- Select contents of the table
SELECT * FROM py_rental_predictions;
```

Копировать

Шаг 4. Прогнозирование с использованием собственной оценки.

В SQL Server 2017 мы представляем встроенную функцию прогнозирования в TSQL. Собственная функция PREDICT позволяет выполнять более быструю оценку с использованием определенных моделей RevoScaleR или revoscalepy с использованием запроса SQL без вызова среды выполнения R или Python. В следующем примере кода показано, как можно обучить модель в Python с помощью функций revoscalepy «Rx», сохранить модель в таблицу в БД и прогнозировать с помощью встроенной оценки.

SQL

```
USE TutorialDB;

--STEP 1 – Setup model table for storing the model
DROP TABLE IF EXISTS rental_models;
GO
CREATE TABLE rental_models (
```

```
        model_name VARCHAR(30) NOT NULL DEFAULT('default
        model'),
        lang VARCHAR(30),
               model VARBINARY(MAX),
               native_model VARBINARY(MAX),
               PRIMARY KEY (model_name, lang)

);
GO

--STEP 2 – Train model using revoscalepy rx_dtree or rxlinmod
DROP PROCEDURE IF EXISTS generate_rental_py_native_model;
go
CREATE PROCEDURE generate_rental_py_native_model (@model_type
varchar(30), @trained_model varbinary(max) OUTPUT)
AS
BEGIN
       EXECUTE sp_execute_external_script
       @language = N'Python'
      , @script = N'
from revoscalepy import rx_lin_mod, rx_serialize_model,
rx_dtree
from pandas import Categorical
import pickle

rental_train_data["Holiday"] = rental_train_data["Holiday"].
astype("category")
rental_train_data["Snow"] = rental_train_data["Snow"].
astype("category")
rental_train_data["WeekDay"] = rental_train_data["WeekDay"].
astype("category")

if model_type == "linear":
       linmod_model = rx_lin_mod("RentalCount ~ Month + Day + Week
       Day + Snow + Holiday", data = rental_train_data)
       trained_model = rx_serialize_model(linmod_model,
       realtime_scoring_only = True);
if model_type == "dtree":
```

```
        dtree_model = rx_dtree("RentalCount ~ Month + Day + WeekDa
        y + Snow + Holiday", data = rental_train_data)
        trained_model = rx_serialize_model(dtree_model,
        realtime_scoring_only = True);
'

       , @input_data_1 = N'select "RentalCount", "Year",
       "Month", "Day", "WeekDay", "Snow", "Holiday" from dbo.
        rental_data where Year < 2015'
       , @input_data_1_name = N'rental_train_data'
       , @params = N'@trained_model varbinary(max) OUTPUT, @
        model_type varchar(30)'
          , @model_type = @model_type
          , @trained_model = @trained_model OUTPUT;
END;
GO

--STEP 3 – Save model to table

--Line of code to empty table with models
--TRUNCATE TABLE rental_models;

--Save Linear model to table
DECLARE @model VARBINARY(MAX);
EXEC generate_rental_py_native_model "linear", @model OUTPUT;
INSERT INTO rental_models (model_name, native_model, lang)
VALUES('linear_model', @model, 'Python');

--Save DTree model to table
DECLARE @model2 VARBINARY(MAX);
EXEC generate_rental_py_native_model "dtree", @model2 OUTPUT;
INSERT INTO rental_models (model_name, native_model, lang)
VALUES('dtree_model', @model2, 'Python');

-- Look at the models in the table
SELECT * FROM rental_models;

GO
```

```
--STEP 4  – Use the native PREDICT (native scoring) to predict
 number of rentals for both models
 DECLARE @model VARBINARY(MAX) = (SELECT TOP(1) native_model
 FROM dbo.rental_models WHERE model_name = 'linear_model' AND
 lang = 'Python');
 SELECT d.*, p.* FROM PREDICT(MODEL = @model, DATA = dbo.rental_
 data AS d) WITH(RentalCount_Pred float) AS p;
 GO

--Native scoring with dtree model
 DECLARE @model VARBINARY(MAX) = (SELECT TOP(1) native_model
 FROM dbo.rental_models WHERE model_name = 'dtree_model' AND
 lang = 'Python');
 SELECT d.*, p.* FROM PREDICT(MODEL = @model, DATA = dbo.rental_
 data AS d) WITH(RentalCount_Pred float) AS p;
 GO
```

Копировать

Мы развернули прогностическую модель в SQL Server с помощью Python.

4.4. Контейнеры для администраторов и разработчиков баз данных

В операционных системах реализуется технология виртуальных машин.

Виртуальные машины — это технология, позволяющая абстрагировать приложения ИС от базовой аппаратной платформы, но они требуют загрузки и запуска всей операционной системы, для того чтобы ИС могла работать.

Виртуальные машины позволяют приложениям ИС запускаться изолированно друг от друга на хост-системе, и для SQL-2019 это было удобным решением для сценариев консолидации (даже несмотря на то, что SQL-2019 допускает размещение нескольких экземпляров на одном компьютере).

Эффективность и возможность решения приложений сквозных технологий цифровой экономики определили реализацию инструмента контейнеров.

Контейнеры можно рассматривать как отдельный уровень абстракции от операционной системы.

Контейнеры не заменяют виртуальные машины. Контейнеры дополняют их.

Фактически одна из самых распространенных сред для запуска контейнеров — это виртуальная машина.

Образ контейнера представляет собой двоичный файл, который описывает набор файлов, организованных в файловой системе, и программу, запускаемую из этих файлов.

Контейнер является экземпляром программы, запускаемой изолированным способом в образе контейнера вместе с файловой системой. Рассмотрим основные свойства контейнеров.

Переносимые — работают везде, где поддерживается Docker (программное обеспечение для автоматизации развертывания и управления приложениями в средах с поддержкой контейнеризации).

Контейнеры являются переносимыми, потому что образ контейнера можно запускать везде, где можно запустить Docker, т. е. практически везде, включая компьютеры, работающие под управлением ОС Windows, Linux и macOS, а также в облачных системах, поддерживающих эти операционные системы или Kubernetes (программная платформа для автоматического управления контейнеризованными приложениями ИС). На практике берем образ контейнера SQL-2019 в виде двоичного файла и помещаем его в любую из систем, где он будет работать точно так же.

Облегченные — пониженное потребление дискового пространства, памяти и ресурсов процессора. Контейнер — это всего лишь процесс, представляющий собой приложение ИС, работающее изолированно. Это делает его намного более легким, чем виртуальная машина, предназначенная для размещения приложения ИС. Требования контейнеров к ресурсам также оптимизированы, потому что, если вы запускаете более одного контейнера из образа, часть файлов образов (называемых читаемым слоем, или слоем, доступным для чтения, о котором мы расскажем далее в этой главе) распределяется между контейнерами. Это уменьшит объем ресурсов, необходимый для запуска нескольких экземпляров SQL-2019 на хост-системе или виртуальной машине.

Согласованные — целостный образ SQL-2019, сценариев и инструментов. Это один из аспектов контейнеров, который помогает решить огромную проблему для SQL-2019. В течение многих лет во многих компаниях сложилась практика, когда администраторы устанавливали отдельные серверы с SQL Server, формируя платформы для тестирования и разработки. Это вызывает сложности, так как несколько разработчиков ИС, использующих один и тот же SQL Server, могут создать множество проблем для администраторов баз данных. Контейнеры предоставляют разработчикам ИС согласованный способ использования SQL-2019, при этом отказавшись от обязательного совместного использования экземпляра SQL-2019. Например, если вы хотите, чтобы все разработчики использовали определенный образ версии SQL Server вместе с определенной базой данных, то можете просто создать образ контейнера. А поскольку контейнеры переносимы, и SQL-2019 теперь работает на Linux, вы можете предоставить один и тот же образ контейнера SQL-2019 для разработчиков ИС, использующих разные платформы. Разработчики macOS могут использовать тот же образ SQL-2019, что и разработчики Windows. На рис. 61 приведены Dev (development) и Ops (operations), показывающие, насколько важна концепция контейнеров для поддержки модели DevOps. DevOps — методология автоматизации технологических процессов сборки, настройки и развертывания программного обеспечения. Методология предполагает активное взаимодействие специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимную интеграцию их технологических процессов друг в друга для обеспечения высокого качества программного продукта. Предназначена для эффективной организации создания и обновления программных продуктов и услуг.

Эффективные — более быстрое развертывание, уменьшение количества исправлений и меньшее время простоя. Контейнеры предоставляют новые эффективные возможности для обновления программного обеспечения, в том числе SQL-2019. Если выполнять обновления для SQL-2019, то контейнеры упрощают эту процедуру, существенно сокращается время простоя, и при необходимости быстрее откатывать обновления.

Рассмотрим схему, показанную на рис. 61, чтобы пояснить, что такое контейнеры и почему они могут решать определенные задачи для современных приложений ИС.

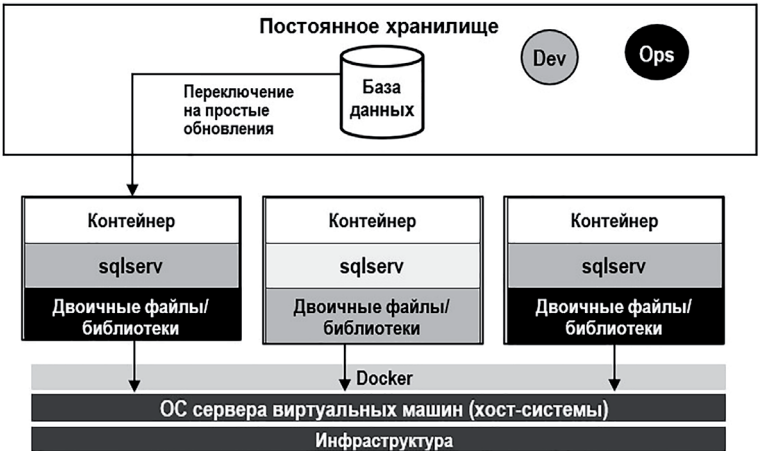

Рис. 61. Концепция контейнеров SQL-2019

На рис. 61 контейнер в середине, выделенный серым цветом, представляет собой остановленный контейнер SQL-2019.

Контейнер слева — это новая версия SQL-2019, которая запускается, но указывает на те же системные и пользовательские базы данных, размещенные в постоянном хранилище.

Черные прямоугольники с надписью «Двоичные файлы/библиотеки» представляют собой двоичные файлы, необходимые для запуска SQL-2019. Они иллюстрируют аспект «облегченности» контейнера, отличающий его от виртуальной машины.

Элемент с надписью Docker обозначает программное обеспечение Docker, которое используется для запуска и управления контейнерами.

Важным являются стрелки, проходящие через Docker к операционной системе хост-узла.

Docker не является слоем между контейнером и операционной системой хост-узла. Другими словами, SQL-2019 не нужно взаимодействовать через некоторый отдельный уровень для выполнения операций ОС ядра. По этой причине контейнеры счи-

таются более легкими — поскольку они содержат программное обеспечение, напрямую взаимодействующее с операционной системой хост-узла.

Уникальной особенностью контейнеров является то, что они работают изолированно друг от друга, отсюда и появился термин «контейнер».

Администратору необходимо понимать, что контейнеры — это программные модули, которые взаимодействуют непосредственно с операционной системой хост-узла.

Операционная система хост-узла может находиться на виртуальной машине или непосредственно на аппаратной платформе.

Поскольку наша программа в контейнере взаимодействует напрямую с операционной системой хост-узла, так же, как и любая другая программа в системе (за исключением того, что, разумеется, контейнеры запускаются специальным изолированным способом), она должна быть скомпилирована и запущена на этой операционной системе.

4.5. Администрирование контейнеров и их применение

Важной составляющей в применении контейнеров является Docker.

При установке Docker в Linux, Windows или macOS устанавливаются следующие компоненты, которые обеспечивают работу контейнеров.

Механизм Docker (docker engine) — состоит из демона Docker (который является «сервисом»), контролирующего все операции по созданию и запуску контейнеров. Механизм Docker поддерживает API (Application programming interface) для программ, взаимодействующих с механизмом построения и запуска контейнеров.

Клиент Docker (docker client) — это приложение, которое использует API механизма Docker для создания и запуска контейнеров. Клиент Docker — это согласованная программа, которая поддерживает все возможности и ведет себя одинаково в Windows, macOS и Linux.

Инструмент для управления набором контейнеров Docker (docker compose) — это приложение, которое позволяет создавать и запускать многоконтейнерные приложения ИС.

Build — команда сборки Docker, которая используется для создания нового образа контейнера. Microsoft создает образы, которые содержат SQL Server, поэтому во многих случаях вам не придется самостоятельно проделывать эту процедуру.

Push — после создания образа нужно будет предоставить возможность другим пользователям. Для отправки или публикации образа контейнера в реестре используется команда docker push. Этот реестр может находиться на локальном сервере или в открытом доступе. Одним из наиболее распространенных реестров в публичном доступе является Docker Hub или hub.docker.com. Microsoft публикует образы своих контейнеров, в том числе и контейнеров с SQL Server, на сайте mcr.microsoft.com (так называемый реестр контейнеров Microsoft).

Pull — любой, кто хочет использовать образ контейнера, должен получить его по запросу, даже если этот образ хранится на локальном сервере. Получить образ контейнера можно с помощью команды docker pull. Механизм Docker (docker engine) будет хранить копию образа локально на хост-узле.

После того как контейнер запущен, им можно управлять.

С помощью **клиента Docker** вы сможете остановить, запустить, перезапустить и удалить контейнер. Кроме того, клиент Docker позволит управлять образами, в том числе удалять их.

Клиент Docker позволяет выполнять мониторинг и управление системой контейнеров. Позволяет получить списки запущенных и остановленных контейнеров, а также выводить данные статистики и журналов для запущенных и остановленных контейнеров.

Клиент Docker позволяет взаимодействовать с запущенными контейнерами, копируя файлы в слой для записи на хост-узле и запуская программу, которая существует в файловой системе контейнеров (которая будет выполняться в том же пространстве имен, что и основная программа-контейнер).

Устанавливать контейнер SQL-2019 можно командами в Dockerfile для SQL-2019:

```
FROM <базовый образ на основе ubuntu или rhel>
LABEL <информация о метке Microsoft>
EXPOSE 1433
COPY библиотеки и двоичные файлы SQL Server>
RUN ./install
CMD ["/opt/mssql/bin/sqlservr"]
```

Команда **FROM** задает базовый образ ОС, на котором построен образ контейнера SQL-2019.

Команда **EXPOSE** позволяет контейнеру SQL-2019 устанавливать соединения, необходимые приложениям для обмена данными, с использованием порта 1433 внутри контейнера. Это важно, поскольку по умолчанию контейнеры изолированы.

Команды **COPY** и **RUN** являются лишь частью процесса сборки для копирования всех двоичных файлов SQL-2019 в файловую систему образа контейнера и установки любых программных зависимостей.

Запускается контейнер следующими командами:

```
docker run
-e 'ACCEPT_EULA=Y' –e 'MSSQL_SA_PASSWORD=Sql2017isfast1
-p 1401:1433
-v sqlvolume:/var/opt/mssql
--hostname sql2019latest
--name sql2019latest
-d
```

Параметры «e» обозначают переменные среды, которые используются для запуска и работы контейнера.

-p 1401:1433

Этот параметр не потребуется, если запускается только один контейнер SQL-2019 на одном хост-узле. Если есть несколько экземпляров SQL-2019, нужно связать порт 1433 с другим портом.

-v sqlvolume:/var/opt/mssql

Данный параметр указывает, какой том следует применять для связывания с каталогом SQL=2019, где хранятся базы данных.

--hostname sql2019latest

Данный параметр необязательный, но он очень удобен. Это удобство заключается в том, что указанное имя вычислительного узла, на котором размещен контейнер, станет значением.

--name sql2019latest

Данный параметр также не является обязательным, но он удобен для управления контейнером. Задав для контейнера имя, можно легко идентифицировать контейнер по имени и управлять им.

-d

Этот параметр определяет, что контейнер будет запускаться в фоновом режиме.

mcr.microsoft.com/mssql/rhel/server:2019-latest

Это тег образа контейнера, который запускается.

Различные примеры подготовки и запуска контейнеров приведены в каталоге **h7_inside_sql_containers\deploy** (открывать браузером Microsoft) или по прямой ссылке: https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-docker-container-deployment?view=sql-server-ver15&pivots=cs1-bash.

Выводы к главе 4

В этой главе мы узнали о новых функциях операционной системы для разработки и эксплуатации ИС. Эти функции обеспечивают высокую эффективность для сквозных технологий цифровой экономики и определяют новые инструменты администрирования в интересах разработчиков программного обеспечения ИС.

Мы продемонстрировали на конкретном примере администрирование операционной системы для поддержки практически любого языка программирования, обновленные источники данных, графовую базу данных, расширения T-SQL и машинное обучение.

Администрирование этих новых функций в сочетании с новыми мощными возможностями интеллектуальной обработки запросов (Intelligent Query Processing) и оптимизацией метаданных tempdb предоставляет необходимый инструментарий для современных разработчиков, специализирующихся на обработке данных ИС. Администрирование службы машинного

обучения (SQL Server Machine Learning Services) помогает усовершенствовать и модернизировать SQL Server, делая его не просто системой управления базами данных, а настоящей платформой данных. SQL-2019 расширяет службы SQL Server ML Services, вводя следующие новые функции:

— внешние библиотеки теперь можно устанавливать для новых пакетов R или Python с помощью оператора T-SQL CREATE EXTERNAL LIBRARY;

— служба Launchpad имеет решающее значение для архитектуры служб SQL Server ML Services. Теперь SQL Server ML Services может быть частью экземпляра всегда включенного отказоустойчивого кластера, включая саму службу Launchpad.

Администрирование использования SQL-2019 в качестве платформы для машинного обучения предоставляет специалисту по анализу данных ИС место для экспериментов и практической работы с рабочими задачами, а администратор и разработчик БД ИС получают контроль за внедрением кода машинного обучения R и Python в Transact-SQL, что способствует эффективному разделению обязанностей.

Мы рассмотрели, что такое контейнеры, и почему с их помощью можно решать современные задачи по разработке приложений ИС. К преимуществам контейнеров относятся их портативность, легкость, надежность и эффективность. Было показано, что контейнеры — это на самом деле просто программы, работающие изолированным и уникальным способом.

Было определено, что программа ИС в контейнере взаимодействует напрямую с операционной системой хост-узла, так же, как и любая другая программа в системе, она должна быть скомпилирована и запущена на этой операционной системе.

Вопросы для самоконтроля

1. Инновационные функции SQL-2019.
2. Графовые базы данных и их применение.
3. В чем различие реляционных баз данных и графовых?
4. Что такое кодировка UTF-8?
5. Службы машинного обучения SQL-2019 (SQL Server Machine Learning Services).

6. Какие языки программирования интегрированы в SQL Server Machine Learning Services?

7. Как связаны язык T-SQL и SQL Server Machine Learning Services?

8. Архитектура служб SQL Server ML Services.

9. Какие действия выполняются при создании прогностической модели с помощью Python и SQL Server ML Services?

10. Зачем необходимо обучение модели?

11. Понятие виртуальных машин в операционной системе.

12. Контейнеры и их отличие от виртуальных машин.

13. Основные свойства контейнеров.

14. Как контейнеры могут решать задачи для современных приложений ИС?

15. Концепция контейнеров SQL-2019.

16. Администрирование контейнеров и Docker.

17. Жизненный цикл контейнера.

18. Основные команды запуска контейнера.

ГЛАВА 5

Сетевое администрирование информационных систем

Главной задачей администрирования является обеспечение надежности, бесперебойности, безопасной работы всей системы с требуемым уровнем производительности.

5.1. Сетевые сервисы

Информационная система обязательно будет базироваться на компьютерной сети, при этом сети бывают условно трех видов:

— локальные (ЛВС, LAN — Local Area Network);
— региональные (РВС, MAN — Metropolitan Area Network);
— глобальные (ГВС, WAN — Wide Area Network).

Сетевой инфраструктуре требуются соответствующие программные приложения или службы, которые должны быть установлены на компьютерах и регулировать трафик данных. В большинстве случаев службы системы доменных имен (DNS) также являются протоколом обмена динамической конфигурации хоста (DHCP) и службы Windows (WINS), которые являются частью базового пакета услуг. Эти приложения должны быть настроены соответствующим образом и постоянно быть доступными.

Активное сетевое оборудование включает в себя такие виды устройств, как повторители (репитеры), мосты, концентраторы, коммутаторы, маршрутизаторы. В корпоративной сети может быть использован богатый набор сетевых протоколов: TCP/IP, SPX/IPX, NetBEUI, AppleTalk и др.

Основу работы сети составляют так называемые сетевые службы (или сервисы).

Среди всего множества сетевых служб следует выделить следующие.

1. Службы сетевой инфраструктуры, обеспечивающие работу в сети TCP/IP.

Служба DHCP по запросу DHCP-клиента выдает ему такие параметры, как уникальный IP-адрес и маска подсети.

Служба DNS выполняет преобразование (разрешение) имен узлов в соответствующие им IP-адреса. Служба DNS была реализована в Интернете в 1981 г., а с 2000 г. (с выходом ОС семейства Windows 2000) она стала основной службой преобразования имен в сетях Microsoft.

Служба WINS регистрирует в сети NetBios имена компьютеров и их IP-адреса, а затем по запросу WINS-клиентов преобразует эти имена в IP-адреса.

2. Файловая служба обеспечивает хранение больших объемов данных и предоставляет к ним доступ пользователей.

3. Служба печати (принт-сервер) предназначена для обеспечения доступа пользователей к одному или нескольким общим принтерам.

4. Служба приложений выполняет задачи обслуживания запросов пользователей на выборку или обработку какой-либо информации.

5. Служба удаленного доступа и серверы VPN (Virtual Private Network — «виртуальная частная сеть») обеспечивают удаленное подключение к локальной сети по модему или через Интернет.

6. Брандмауэры (межсетевые экраны) используются при подключении к Интернету для защиты внутренней сети от проникновения или атаки злоумышленников на корпоративные серверы. Прокси-серверы (серверы-посредники) выполняют функции контроля доступа пользователей в Интернет и кэширования часто запрашиваемых веб-страниц (что позволяет снизить расходы на пользование Интернетом).

7. Службы Web и FTP предоставляют для внешних (а часто — и для внутренних) пользователей доступ к Web- и FTP-ресурсам, размещенным в данной сети.

Windows Server предоставляет системному администратору широкий набор инструментов для решения задач управления. Основными из этих инструментов являются следующие:

— консоль управления (Microsoft Management Console, MMC);

— мастера (Wizards);
— утилиты командной строки.

5.2. Технология виртуализации данных (RAID-массивы)

RAID (Redundant Array of Independent Disks) — позволяет превратить несколько дисковых накопителей в один большой и быстрый диск. Его можно использовать в качестве хранилища данных с функцией автоматического резервного копирования или настроить как системный диск повышенной отказоустойчивости.

В основе теории RAID лежат пять основных принципов.

1. **Массив** — объединение нескольких физических или виртуальных накопителей в один большой диск с возможностью единой настройки, форматирования и управления.

2. **Метод зеркалирования** — способ повысить надежность хранения информации через создание копии исходного диска на другом носителе, входящем в массив.

3. **Дуплекс** — один из методов зеркалирования, в котором используется вдвое большее количество накопителей для создания копий.

4. **Чередование** — увеличение производительности диска благодаря блочной разбивке данных при записи.

5. **Четность** — технология, сочетающая в себе чередование и зеркалирования.

Определено несколько типов RAID.

Основные отличия между конфигурациями или уровнями RAID заключаются в методах формирования и размещения данных, а также в алгоритмах распределения информации на носителях.

Базовые типы RAID-массивов — RAID 0 и RAID 1 (рис. 62 и 63). Остальные уровни считаются их производными, сочетающими в себе достоинства той или иной базовой модели.

Технология виртуализации RAID 0 называется striping («чередование»). Для ее реализации применяется от двух до четырех накопителей, которые совместно выполняют процедуру «чтения/записи».

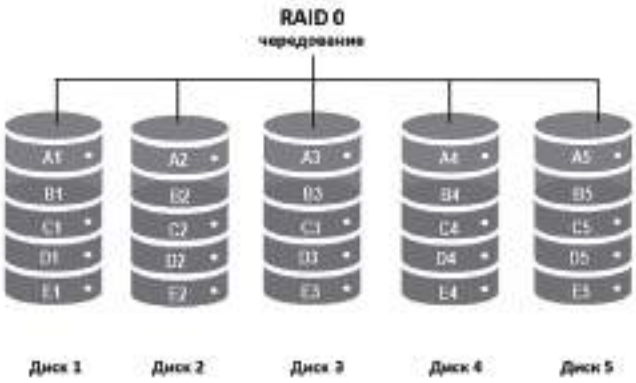

Рис. 62. RAID 0 — чередование

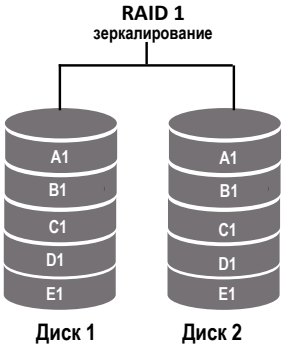

Рис. 63. RAID 1 — зеркалирование

При записи информация разделяется на блоки, которые одновременно сохраняются на накопители.

Достоинства:

— дисковый RAID-массив уровня 0 обеспечивает ощутимый прирост скорости, который прямо пропорционально зависит от кратности количества накопителей;

— использование всего дискового пространства, т. е. при установке четырех дисков по 2 ТБ общий объем RAID-массива составит 8 ТБ.

Недостатки:

— нарушение отказоустойчивости. Иногда возможен отказ в операциях чтения или записи;

— при выходе из строя одного накопителя информация полностью теряется.

Использование. Применяется в приложениях для скоростного обмена информацией, в хранилищах временных файлов. Также RAID 0 нужен для систем, использующих некритичные по важности массивы данных.

Технология RAID 1 называется mirroring («зеркалирование»). Она подразумевает использование от двух до четырех накопителей. Однако при этом теряется половина объема дисков, поскольку это пространство используется резервированием данных.

Проще, если RAID-система состоит из двух жестких дисков, тогда при выходе одного из них информация не потеряется полностью, поскольку один накопитель является точной копией другого.

Достоинства:

— надежность хранения информации;

— простота реализации;

— высокая производительность при выполнении операции чтения;

— минимальная комплектация составляет всего два жестких диска.

Недостатки:

— низкая производительность;

— емкость RAID-массива делится на 2, что обусловлено резервированием информации;

— замена неисправного накопителя требует полного отключения системы.

Использование. Уровень RAID 1 необходимо применять для увеличения надежности хранения информации на серверах.

Технология RAID 5 — «чередование с четностью» (рис. 64) считается наиболее распространенной и безопасной. Для подобной конфигурации необходимо минимум три диска, а максимальное допустимое количество — 16.

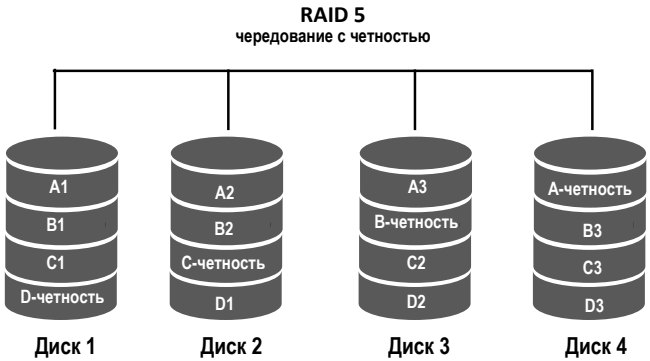

Рис. 64. RAID 5 — чередование с четностью

При записи информации происходит разделение на блоки данных, но с одним условием — на один из дисков, называемый блок «четность данных» **(Parity Drive, PD)**, происходит запись информации для восстановления. Этот подход позволяет спасти данные при повреждении одного из накопителей.

RAID 5 может реализовываться программным методом при помощи специальных утилит, но IT-специалисты рекомендуют все же отдать предпочтение аппаратному способу.

Достоинства:

— увеличена скорость чтения за счет одновременной обработки данных с нескольких независимых потоков от дисков массива;

— информация не «потеряется» при повреждении одного накопителя;

— при замене неисправного диска происходит автоматическое восстановление информации.

Недостатки:

— иногда происходят отказы дисков;

— если объем поврежденного накопителя 4 ТБ и более, при замене его на идентичный диск, восстановление может занять более одного дня;

— если диск «четности» вышел из строя при выполнении процедуры восстановления, то информация будет окончательно утеряна;

— минимальное количество накопителей — три.

Использование. Технология виртуализации 5-го уровня (RAID 5) прекрасно подойдет для безопасного хранения данных, но при этом не будет утрачена производительность. Очень часто ее используют файловые серверы.

Технология виртуализации 6-го уровня («чередование с двойной четностью») похожа на RAID 5 (рис. 65). Отличие состоит в записи информации для восстановления на два диска. Первый — блок «четность данных» **(PD)** — используется в архитектуре RAID 5 для резервного хранения данных. Второй диск «четности» дублирует работу первого. Его работа основана на коде Рида-Соломона (Reed-Solomon), поэтому диск часто имеет краткое обозначение — **RS** или **Q.**

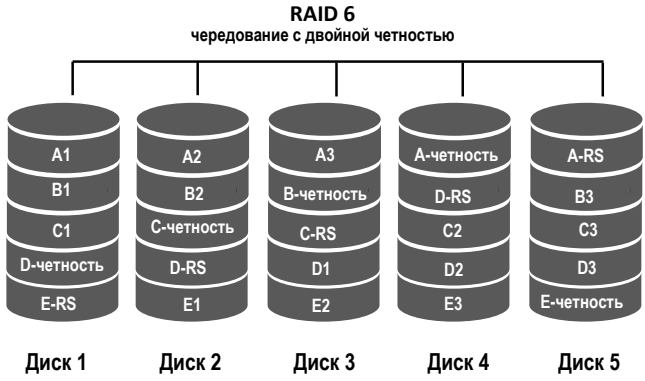

Рис. 65. RAID 6 — чередование с двойной четностью

Благодаря использованию принципа двойной четности система может перенести без потерь информации отказ сразу двух жестких дисков. Однако для создания RAID 6 потребуется минимум четыре накопителя.

Достоинства:

— высокая скорость считывания и записи данных;

— поддержка двух одновременно вышедших из строя накопителей.

Недостатки:

— время на операцию записи на 20 % больше, чем для RAID 5;

— минимальная вероятность отказа дисков;

— восстановление после сбоя занимает много времени;
— для реализации необходимо четыре накопителя.

Использование. RAID 6 является более надежной конфигурацией, чем RAID 5-го уровня. Она часто применяется на файловых серверах, где используются большие объемы данных.

Технология виртуализации 10 — «гибрид» RAID нулевого и 1-го уровней, сочетающая в себе все их преимущества (рис. 66).

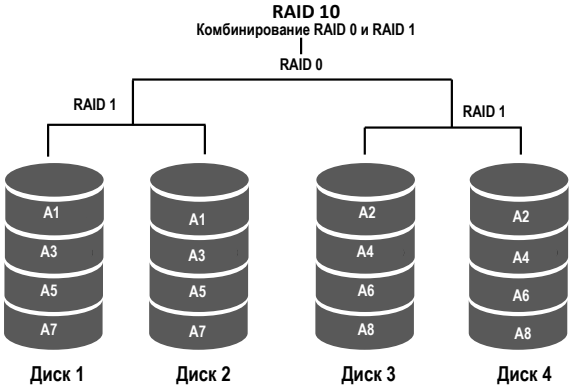

Рис. 66. RAID 10 комбинирование RAID 0 и RAID 1

Достоинства:
— высокая скорость восстановления данных;
— высокая надежность;
— быстродействие.

Недостатки:
— дороговизна реализации;
— емкость, уходящая на зеркалирование, эквивалентна 50 % от всего объема дисков.

Использование. Гибридная технология RAID 10 используется в тех же случаях, что и RAID 0 и RAID 1. Для создания используется утилита RAID.

5.3. IP-адресация и маршрутизация в компьютерных сетях

Под понятием «Сеть» подразумевается физическая связь между компьютерами, осуществляющими пересылку пакетов, с единой системой адресации.

Адресация — процесс, при котором микропроцессор обращается к определенному сегменту памяти или внешнему устройству, используя определенные адреса.

Маршрутизация — это процесс направления пакета по лабиринту сетей, находящихся между источником и адресатом.

Самым распространенным «семейством» протоколов, на котором построен Интернет, является TCP/IP — это ряд протоколов под общим названием TCP/IP (по названиям двух основных протоколов: TCP и IP).

Каждый компьютер в сетях TCP/IP имеет **адреса трех уровней:** физический (MAC-адрес), сетевой (IP-адрес) и символьный (DNS — имя).

Адрес IP представляет собой 32-разрядное двоичное число, разделенное на группы по 8 бит, называемые **октетами.** Например, 00010001 11101111 00101111 01011110.

Обычно IP-адреса записываются в виде четырех десятичных октетов и разделяются точками. Таким образом, приведенный выше IP-адрес можно записать в следующей форме: 17.239.47.94.

IP-адрес состоит из двух логических частей — ***номера подсети*** (ID подсети) и ***номера узла*** (ID хоста) в этой подсети. При передаче пакета из одной подсети в другую используется ID подсети. Когда пакет попал в подсеть назначения, ID хоста указывает на конкретный узел в рамках этой подсети.

Определение того, какая часть адреса отводится под номер подсети, осуществляется двумя способами — с помощью классов и с помощью масок.

В схеме классовой адресации существует **пять классов,** основными являются классы A, B, C.

Классы существенно отличаются друг от друга по размерам и сложности. Они определяют, сколько бит в IP-адресе отводится под номер сети и сколько — под номер узла.

Класс A. Сеть класса A имеет адреса, которые начинаются с числа от 1 до 127 для первого октета, а остальная часть адре-

са — это адрес узла. Таким образом, класс A допускает максимум 126 сетей, а в каждой из них — до 16 777 214 компьютеров. Как правило, это сети огромных компаний, которых в мире немного, объединяющих большое число сетевых устройств.

Класс B. В сети класса B для описания адреса сети используется первые два октета, а остальная часть — это адреса узлов. Первый октет принимает значения от 128 до 191, что дает максимум 16 384 сети, в каждой из которых — до 65 534 узлов. Адреса класса B назначаются сетям большого и среднего размера.

Класс C. Адреса сетей класса C начинаются с числа от 192 до 223 и используют три первых октета для описания адреса сети. Последний октет обозначает адрес узла. Таким образом, класс C допускает максимум 2 097 152 сети, по 254 компьютера в каждой. Адреса этого класса назначают малым сетям.

Чтобы записать ID подсети, в поле номера узла в IP-адресе ставят нули. Чтобы записать ID хоста, в поле номера подсети ставят единицы. Например, если в IP-адресе 172.16.123.1 первые два байта отводятся под номер подсети, остальные два байта — под номер узла, то номера записываются следующим образом: ID подсети — 172.16.0.0; ID хоста — 0.0.123.1.

Общее правило: под ID подсети отводятся *первые* несколько бит IP-адреса, оставшиеся биты обозначают ID хоста.

Для большей гибкости IP-адреса применяется маска подсети.

Маска подсети (subnet mask) — это число, которое используется в паре с IP-адресом; двоичная запись маски содержит единицы в тех разрядах, которые должны в IP-адресе интерпретироваться как номер сети.

Маска подсети записывается либо в виде, аналогичном записи IP-адреса, например, 255.255.255.0, либо совместно с IP-адресом с помощью указания числа единичных разрядов в записи маски, например, 192.168.1.1/24, т. е. в маске содержится 24 ед. (255.255.255.0).

Рассмотрим пример расчета количества подсетей и хостов для сети с использованием формулы Cisco.

Cisco-формула расчета сетей:

$$K = 2^n,$$

где K — количество подсетей; n — количество занятых бит от порции хоста.

Cisco формула расчета хостов (узлов):

$$Kh = 2^n - 2,$$

где *Kh* — количество хостов в подсети; *n* — это количество свободных бит (нулей) в порции хоста; 2 — адрес сети (в порции хоста все нули) и широковещательный адрес (в порции хоста все единицы).

Объяснение формул расчета сетей IP-адресов

IP-адрес состоит из 32 битов, которые поделены на 4 части по 8 бит соответственно (эти части называются октетами). В жизни используется запись IP-адреса в десятичном виде.

Примеры IP-адресов:

— 172.16.2.15 = 10101100.00010000.00000010.00001111;

— 178.68.128.168 = 10110010.01000100.10000000.10101000;

— 217.20.147.94 = 11011001.00010100.10010011.01011110.

Из этих 32 битов часть относится к адресу хоста, которому принадлежит этот IP-адрес, а другая часть относится к адресу сети, в которой находится этот хост. Первая часть (слева направо) IP-адреса обозначает адрес сети, а вторая часть (оставшиеся биты) — адрес хоста. Чтобы узнать, сколько битов относится к адресу сети, надо воспользоваться маской сети.

Маска сети тоже состоит из 32 битов, но в отличие от IP-адреса, в маске единицы и нолики не могут перемешиваться. В жизни используется запись сетевой маски в десятичном виде.

Примеры масок сети:

— 255.255.255.0 = 11111111.11111111.11111111.00000000;

— 255.0.0.0 = 11111111.00000000.00000000.00000000;

— 255.255.240.0 = 11111111.11111111.11110000.00000000;

— 255.255.255.128 = 11111111.11111111.11111111.10000000.

Еще чаще маска сети записывается в виде короткого **префикса маски.** Число в префиксе обозначает количество бит, относящихся к адресу сети:

— /16 = 11111111.11111111.00000000.00000000 = 255.255.0.0;

— /24 = 11111111.11111111.11111111.00000000 = 255.255.255.0;

— /26 = 11111111.11111111.11111111.11000000 = 255.255.255.192.

Чтобы узнать, какая часть IP-адреса относится к порции сети, необходимо выполнить бинарную логическую операцию **AND (И).**

Смысл операции заключается в сравнении двух битов, причем только в одном случае бинарная операция дает единицу на выходе — в случае сравнения двух единиц. В остальных случаях логическая операция AND дает на выходе 0.

Результаты сравнения логической операцией AND двух битов:

— **1 AND 1 = 1;**
— **1 AND 0 = 0;**
— **0 AND 1 = 0;**
— **0 AND 0 = 0.**

Операция AND над IP-адресом и маской

Представим, что у нас есть IP-адрес 192.168.1.31 с маской сети в виде префикса /24, наша задача вычислить адрес сети, порцию сети, порцию хоста.

Сначала надо перевести IP-адрес из десятичной системы счисления в двоичную систему. Затем перевести префикс в двоичный вид и нормальный вид маски сети (десятичный). Далее остается только сложить IP-адрес (192.168.1.31) с маской (/24) с помощью логической операции AND.

192.168.1.31/24
192.168.1.31 = 11000000.10101000.00000001.00011111
/24 = 11111111.11111111.11111111.00000000 = 255.255.255.0.

Таким образом будет выглядеть адрес сети в двоичном коде:

11000000.10101000.00000001.0001111 AND
1111111.11111111.11111111.00000000 =
=11000000.10101000.00000001.00000000
192.168.1.0/24

Вот мы и узнали адрес сети в десятичном виде с сетевым префиксом. Единицы в маске указывают на длину порции адреса сети (11000000.10101000.00000001), а нули — на порцию адреса хоста (.00011111).

Примеры расчета сетей

Деление сети осуществляется присвоением битов из порции адреса хоста к порции адреса сети. Тем самым мы увеличиваем возможное количество подсетей, но уменьшаем количество хостов в подсетях. Чтобы узнать, сколько получается подсетей из присвоенных битов, надо воспользоваться cisco-формулой расчета сетей: 2^n, где n является количеством присвоенных бит.

Пример расчета сети на 2 подсети

У нас есть адрес сети 192.168.1.0/24, нам надо разделить имеющуюся сеть на 2 подсети. Попробуем забрать от порции хоста 1 бит и воспользоваться формулой: $2^1 = 2$; это значит, что если мы заберем один бит от части хоста, то мы получим 2 подсети. Присвоение одного бита из порции хоста увеличит префикс на один бит: /25. Теперь надо выписать два одинаковых IP-адреса сети в двоичном виде, изменив только присвоенный бит (у первой подсети присвоенный бит будет равен 0, а у второй подсети — 1). Захваченный бит выделим жирным шрифтом.

Две подсети (захваченный бит выделим жирным шрифтом):

1) 11000000.10101000.00000001.**0**0000000;

2) 11000000.10101000.00000001.**1**0000000.

Теперь запишем рядом с двоичным видом десятичный, и добавим новый префикс. Жирным шрифтом помечена порция подсети, а обычным — порция хоста:

1) **11000000.10101000.00000001.**00000000 = 192.168.1.0/25;

2) **11000000.10101000.00000001.1**0000000 = 192.168.1.128/25.

Все, сеть разделена на 2 подсети. Как мы видим выше, порция хоста теперь составляет 7 бит.

Чтобы высчитать, сколько адресов хостов можно получить, используя 7 бит, необходимо воспользоваться cisco-формулой расчета хостов:

$$2^n - 2,$$

где n = количество бит в порции хоста.

$$2^7 - 2 = 126 \text{ хостов.}$$

Вычитаемая цифра «2» является двумя адресами, которые нельзя присвоить хосту: адрес сети и широковещательный адрес.

Адрес сети — это когда в порции хоста все нули, а широковещательный адрес — это когда в порции хоста все единицы. Выпишем эти адреса для каждой подсети в двоичном и десятичном виде:

— **11000000.10101000.00000001.0**0000000 = 192.168.1.0/25 (адрес сети первой подсети);

— **11000000.10101000.00000001.0**1111111 = 192.168.1.127/25 (широковещательный адрес первой подсети);

— 11000000.10101000.00000001.10000000 = 192.168.1.128/25 (адрес сети второй подсети);

— **11000000.10101000.00000001.1**1111111 = 192.168.1.255/25 (широковещательный адрес второй подсети).

Некоторые IP-адреса являются особыми, они не должны применяться для идентификации обычных сетей.

1. Если первый октет ID-сети начинается с 127, такой адрес считается адресом машины — источника пакета. В этом случае пакет не выходит в сеть, а возвращается на компьютер-отправитель. Такие адреса называются loopback («петля», «замыкание на себя») и используются для проверки функционирования стека TCP/IP без реальной отправки пакета по сети.

2. Если все биты IP-адреса равны нулю, адрес обозначает узел-отправитель и используется в некоторых сообщениях ICMP (например, на начальном этапе получения IP-адреса от DHCP-сервера).

3. Если все биты IP-адреса равны 1, адрес называется ограниченным широковещательным (limited broadcast). Пакеты, направленные по такому адресу, рассылаются всем узлам той подсети, в которой находится отправитель пакета. В данном случае пакет с таким адресом не выходит за пределы своей сети, так как маршрутизатор, связывающий сети, его дальше «не пропустит». Такой особый IP-адрес (limited broadcast) может иметь также другой вид — все биты Network ID будут равны 0, а все биты Host ID — равны 1. Пакет с таким адресом также не выйдет за пределы сети отправителя, так как нули в идентификаторе сети указывают именно на сеть, где находится отправитель пакета.

4. Если все биты ID хоста равны 1, а биты Network ID идентифицируют определенную сеть (но не ту, где находится отправитель пакета), то адрес называется широковещательным (broadcast); пакеты, имеющие широковещательный адрес, доставляются всем узлам подсети назначения. В случае, если сеть отправителя и сеть назначения совпадают, то адрес принято называть также broadcast, хотя фактически пакет не выйдет за пределы сети отправителя.

5. Если все биты ID хоста равны 0, адрес считается идентификатором подсети (subnet ID).

5.4. Протокол IPv6

Основная проблема протокола Ipv4 — это дефицит адресов в сети Интернет.

В результате была разработана новая система протоколирования сетевого взаимодействия — интернет-протокол IPv6 (Internet Protocol version 6). Однако массовый переход на более прогрессивную технологию обусловлен некоторыми сложностями.

5.5. Понятие маршрутизации. Таблицы маршрутизации

Маршрутизация — это процесс определения пути следования информации в сетях связи. Маршрутизация служит для приема пакета от одного устройства и передачи его другому устройству через другие сети.

Маршрутизатором или шлюзом называется узел сети с несколькими интерфейсами, каждый из которых имеет свой MAC-адрес и IP-адрес.

Виды маршрутизации

1. Прямая маршрутизация. При прямой маршрутизации отправитель в определенной IP-сети может напрямую передавать кадры любому получателю в той же сети. При этом не требуется функциональность IP-маршрутизации.

Например, узлу 10.1.1.1 необходимо передать пакет узлу 10.2.2.2. Первое, что он делает — определяет, находится ли IP-адрес получателя в одной с ним сети.

Для этого сравнивает свой номер сети 10 с номером сети получателя 10. Делает вывод, что узел-получатель находится в одном с ним сегменте сети.

С помощью протокола ARP определяет MAC-адрес узла-получателя и посылает пакет по этому адресу.

ARP (англ. Address Resolution Protocol — протокол определения адреса) — протокол в компьютерных сетях, предназначенный для определения MAC-адреса, имея IP-адрес другого компьютера.

2. Косвенная маршрутизация. Косвенная маршрутизация происходит в том случае, если отправитель и получатель находятся в разных IP-сетях. Косвенная маршрутизация требует, что-

бы отправитель передавал пакеты маршрутизатору для доставки их через распределенную сеть.

Например, узел 10.1.1.1 имеет пакет, который нужно отправить узлу 172.16.0.1.

1. Узел назначения находится не на одной с передающим узлом сети. Узел 10.1.1.1 сконфигурирован так, что любые пакеты, требующие косвенной маршрутизации, передаются его шлюзу по умолчанию — маршрутизатору 1.

2. Чтобы доставить пакет маршрутизатору 1, узлу 10.1.1.1 необходим MAC-адрес маршрутизатора 10.3.3.3. Если MAC-адрес узлу 10.1.1.1 неизвестен, он отправляет ARP-запрос, чтобы его получить. Затем пакет, предназначенный для 172.16.0.1, отправляется маршрутизатору 1.

3. Маршрутизатор 1 осознает, что он подсоединен к сети 172.16. и полагает, что узел 172.16.0.1 должен быть частью этой сети. Маршрутизатор 1 реализует свою собственную процедуру прямой маршрутизации и посылает ARP-запрос, ища узел назначения.

Таблица маршрутизации — это база данных, хранящаяся на маршрутизаторе, которая описывает соответствие между адресами назначения и интерфейсами, через которые следует отправить пакет данных до следующего узла.

Таблица маршрутизации содержит: адрес узла назначения, маску сети назначения, адрес шлюза, интерфейс, метрика (табл. 1).

Таблица 1

Пример таблицы маршрутизации

Сетевой адрес	Маска	Адрес шлюза	Интерфейс	Метрика
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1
0.0.0.0	0.0.0.0	198.21.17.7	198.21.17.5	1
56.0.0.0	255.0.0.0	213.34.12.4	213.34.12.3	15
116.0.0.0	255.0.0.0	213.34.12.4	213.34.12.3	13
123.13.0.0	255.255.0.0	198.21.17.6	198.21.17.5	2
198.21.17.0	255.255.255.0	198.21.17.5	198.21.17.5	1
198.21.17.5	255.255.255.255	127.0.0.1	127.0.0.1	1

Адрес шлюза обозначает адрес маршрутизатора в сети, на который необходимо отправить пакет, следующий до указанного адреса назначения.

Интерфейс — физический порт, через который передается пакет.

Метрика — числовой показатель, задающий приоритет маршрута.

Маска подсети — битовая маска для определения по IP-адресу адреса подсети и адреса узла этой подсети. В отличие от IP-адреса маска подсети не является частью IP-пакета.

Сетевой адрес — идентификатор устройства, работающего в компьютерной сети.

Способы размещения записей в таблицу

Размещение записей в таблице маршрутизации может производиться тремя различными способами.

Первый способ предполагает применение прямого соединения, при котором маршрутизатор сам определяет подключенную подсеть.

Прямой маршрут — это маршрут, который является локальным по отношению к маршрутизатору.

Второй способ предполагает занесение маршрутов вручную. В данном случае имеет место статическая маршрутизация.

Третий способ подразумевает автоматическое размещение записей с помощью протоколов маршрутизации. Данный способ называется динамической маршрутизацией.

Расчет метрики

В качестве параметров для расчета метрик могут выступать:

1) ширина полосы пропускания;

2) задержка (время для перемещения пакета от источника к получателю);

3) загрузка (загруженность канала в единицу времени);

4) надежность (относительное количество ошибок в канале);

5) количество хопов (переходов между маршрутизаторами).

Если маршрутизатору известно более одного маршрута до сети получателя, то он сравнивает метрики этих маршрутов и передает в таблицу маршрутизации маршрут с наименьшей метрикой (стоимостью).

Команда Route

Команда Route выводит на экран все содержимое таблицы IP-маршрутизации и изменяет записи. Запущенная без параметров, команда route выводит справку. Рассмотрим некоторые примеры команды route в командной строке Windows:

— чтобы вывести на экран все содержимое таблицы IP-маршрутизации, введите команду: **route print;**

— чтобы вывести на экран маршруты из таблицы IP-маршрутизации, которые начинаются с 10., введите команду: **route print 10.*;**

— чтобы добавить маршрут по умолчанию с адресом стандартного шлюза 192.168.12.1, введите команду: **route add 0.0.0.0 mask 0.0.0.0 192.168.12.1;**

— чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1, введите команду: **route add 10.41.0.0 mask 255.255.0.0 10.27.0.1;**

— чтобы добавить постоянный маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1, введите команду: **route -p add 10.41.0.0 mask 255.255.0.0 10.27.0.1.**

Многие маршрутизаторы могут маршрутизировать TCP/IP, IPX и AppleTalk. Но поскольку работа Windows Server и интернета основывается на TCP/IP, основное внимание уделяется маршрутизации TCP/IP.

При использовании TCP/IP-адрес сети определяется IP-адресом в сочетании с маской подсети. Адрес сети идентифицирует сеть, где находится данное устройство.

Между разъединенными сетями может требоваться обмен информацией, и тогда на помощь приходит маршрутизация.

Маршрутизация — это процесс передачи информации через межсетевую границу.

Точка отправки называется *источником (отправителем),* а точка приема — *пунктом назначения (получателем).*

Промежуточное устройство (обычно маршрутизатор, иногда это несколько устройств) отвечает за передачу информации из одной сети в другую, пока эта информация не дойдет до указанного получателя.

Например, когда компьютер одной сети отправляет информацию компьютеру, который находится в другой сети, он направляет эту информацию маршрутизатору. Маршрутизатор рассматривает этот пакет и использует адрес получателя в заголовке пакета для передачи информации в соответствующую сеть.

Таблицу маршрутизации можно просмотреть путем выполнения команды route print (рис. 67).

Согласно примеру, представленному на рис. 67 (результат использования команды route print), можно увидеть, что таблицы разделены на пять колонок. Первой идет колонка сетей. В ней представлены все сетевые сегменты, к которым подключен маршрутизатор.

Колонка Netmask показывает маску подсети, но не сетевого интерфейса, к которому подключен сегмент, а самого сегмента. Это позволяет маршрутизатору определить класс адреса для сети места назначения.

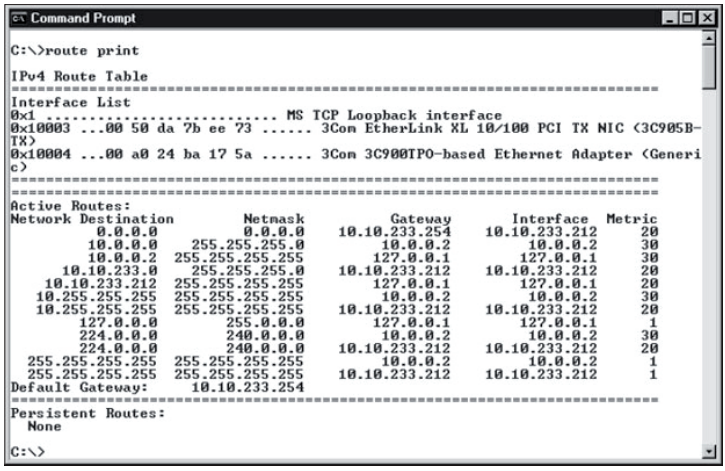

Рис. 67. Пример таблицы маршрутизации

Третьей является колонка шлюза. После того как маршрутизатор определил сеть назначения, в которую необходимо отправить пакет, он сверяется со списком шлюза.

Данный список «говорит» маршрутизатору, через какой IP-адрес необходимо отправлять пакет в сеть назначения.

Существует множество других вариантов использования команды ROUTE. Ее синтаксис следующий:

route [**-f**] [**-p**] [**command** [**destination**] []

Переключатель *-f* является необязательным. Он указывает Windows на необходимость очистить таблицы маршрутизации от пунктов шлюза. Если данный переключатель используется совместно с другими командами, то пункты шлюза будут удалены перед выполнением других инструкций, содержащихся в команде.

Переключатель *-p* делает определенный маршрут постоянным. Обычно при перезагрузке сервера любые определенные через команду ROUTE маршруты удаляются. Переключатель *-p* указывает на необходимость сохранять данный маршрут даже при перезагрузке системы.

Командная часть в синтаксисе ROUTE достаточно проста. Она может состоять из четырех вариантов: PRINT, ADD, DELETE и CHANGE. Пример команды ROUTE PRINT приведен выше (рис. 67), но и у нее могут быть варианты. Например, можно использовать специальные символы в команде. Если нужно напечатать маршруты для подсети 192.x.x.x, можно воспользоваться командой ROUTE PRINT 192*.

Команда ROUTE DELETE работает так же, как и ROUTE Print. Просто вводится ROUTE DELETE, а следом место назначения или шлюз, который необходимо удалить из таблицы маршрутизации. Например, при желании удалить шлюз 192.0.0.0 введите ROUTE DELETE 192.0.0.0.

Все вышесказанное касается и команд ROUTE CHANGE и ROUTE ADD. При введении данной команды следует определить место назначения, маску подсети и шлюз. Также можно указать метрики и интерфейс. Далее рассмотрим примеры использования команды route.

1. Чтобы добавить маршрут по умолчанию с адресом стандартного шлюза 192.168.12.1, введите команду:

route add 0.0.0.0 mask 0.0.0.0 192.168.12.1.

2. Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1, введите команду:

route add 10.41.0.0 mask 255.255.0.0 10.27.0.1.

3. Чтобы добавить постоянный маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0 и следующим адресом перехода 10.27.0.1, введите команду:

route -p add 10.41.0.0 mask 255.255.0.0 10.27.0.1.

4. Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0, следующим адресом перехода 10.27.0.1 и метрикой стоимости 7, введите команду:

route add 10.41.0.0 mask 255.255.0.0 10.27.0.1 metric 7.

5. Чтобы добавить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0, следующим адресом перехода 10.27.0.1 и использованием индекса интерфейса 0x3, введите команду:

route add 10.41.0.0 mask 255.255.0.0 10.27.0.1 if 0x3.

6. Чтобы удалить маршрут к конечной точке 10.41.0.0 с маской подсети 255.255.0.0, введите команду:

route delete 10.41.0.0 mask 255.255.0.0.

7. Чтобы удалить все маршруты из таблицы IP-маршрутизации, которые начинаются с 10., введите команду:

route delete 10.*.

8. Чтобы изменить следующий адрес перехода для маршрута с конечной точкой 10.41.0.0 и маской подсети 255.255.0.0 с 10.27.0.1 на 10.27.0.25, введите команду:

route change 10.41.0.0 mask 255.255.0.0 10.27.0.25.

Маршрутизаторы также могут использовать сообщения об изменениях маршрутной информации для взаимодействия с другими маршрутизаторами, это позволяет маршрутизаторам сравнивать и обновлять свои таблицы маршрутизации, вводя в них информацию о маршрутах в другие сети.

Статическая маршрутизация

Задает единственный путь, который должен использоваться для передачи информации между двумя точками. Администратор должен задавать и конфигурировать статические маршруты в таблицах маршрутизации, и они не изменяются, пока это не сделает администратор. Сетевые среды со статической маршрутизацией организуются достаточно просто и особенно подходят для небольших окружений, где возможны лишь небольшие изменения в топологии маршрутизации.

Основным недостатком сетевых сред со статической маршрутизацией является то, что они не адаптируются к изменению

состояния сети. Например, в случае отключения маршрутизатора или канала статический маршрут не позволяет перенаправлять пакеты на другие маршрутизаторы, чтобы передать их нужным получателям. Кроме того, при добавлении или удалении какой-либо сети в вашем окружении администратор должен задавать возможные сценарии маршрутизации и конфигурировать их соответствующим образом. Поэтому сетевые среды со статической маршрутизацией (в особенности те, что подвержены частым изменениям) не подходят для более крупных сетей. Достаточно оценить расходы на администрирование, чтобы понять, что статическая маршрутизация подходит только для небольших сетевых окружений.

В качестве эмпирического правила используйте статические маршруты только при следующих условиях:

— сетевые окружения с небольшим числом сетей;

— соединения, которые не предполагается изменять в ближайшем будущем (например, маршрутизатор, который используется как последнее средство, когда информацию нельзя маршрутизировать иным способом).

Авто-статическая маршрутизация

Маршрутизаторы на базе Windows Server, которые используют статические маршруты, могут иметь собственные таблицы маршрутизации, обновляемые вручную или автоматически. Автоматически обновляемые статические маршруты называют авто-статической маршрутизацией. Соответствующие обновления можно конфигурировать с помощью интерфейса RRAS или с помощью утилиты NETSH. Это позволяет обновлять информацию маршрутизаторов Windows Server только в определенные периоды времени, что дает «экономию затрат» на соединения и использование меньшей доли пропускной способности каналов.

Авто-статические обновления поддерживаются только в тех случаях, когда вы используете протокол RIP для IP.

Динамическая маршрутизация

Как можно понять из названия, алгоритмы динамической маршрутизации адаптируются к изменениям сетевой среды без ручного вмешательства. Вносимые изменения почти мгновенно отражаются в информации маршрутизатора. Условия, при которых целесообразно использовать динамическую маршрутизацию:

— происходит отключение маршрутизатора или канала, что требует изменения маршрута для передаваемой информации;

— в интерсети добавляется или удаляется маршрутизатор;

— большое сетевое окружение, где имеется много сценариев маршрутизации;

— большое сетевое окружение, в котором часто происходят изменения сетевой топологии.

Маршрутизация с коммутируемым соединением по требованию (demand/dial routing)

Большинство протоколов маршрутизации (RIP, OSPF и т. д.), которые используют для взаимодействия с другими маршрутизаторами, периодически отправляют маршрутную информацию, чтобы адаптироваться к динамическим изменениям состояния сети. Это требуется для того, чтобы информация передавалась по маршрутам с наименьшей «стоимостью». Однако существуют ситуации, когда периодические обновления маршрутизаторов весьма нежелательны. В таких ситуациях можно использовать маршрутизацию с коммутируемым соединением по требованию.

5.6. Протокол DHCP

Протокол **DHCP (Dynamic Host Configuration Protocol)** — это стандартный протокол, определяемый RFC 2131, позволяющий серверу динамически распределять IP-адреса и сведения о конфигурации клиентам. Как правило, DHCP-сервер предоставляет клиенту по крайней мере сведения об IP-адресе.

DHCP работает по модели «клиент — сервер». Он автоматически раздает IP-адреса и другие параметры конфигурации устройствам, чтобы те могли работать в сети.

Клиент — это устройство, которому надо получить IP-адрес для работы в сети. Это может быть телефон, планшет, ноутбук или компьютер.

Сервер — это устройство, которое раздает IP-адреса клиентам и следит за тем, чтобы два клиента не получили одинаковый IP-адрес.

Сервер и клиент обмениваются сообщениями по принципу «запрос-ответ». Взаимодействие состоит из четырех этапов и со-

кращенно называется «DORA» (рис. 68). По одной букве на каждый этап:

1) поиск — Discover;
2) предложение — Offer;
3) запрос — Request;
4) подтверждение — Acknowledgement (ACK).

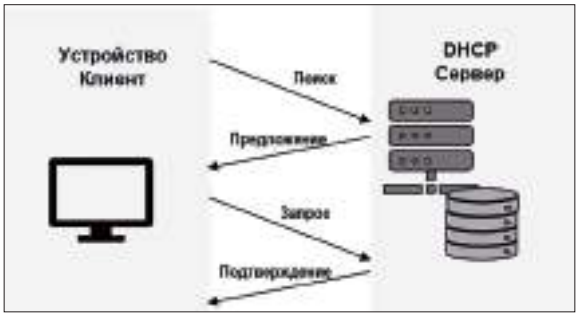

Рис. 68. Обмен «запрос-ответ»

Поиск (Discover): Клиент → Сервер

На этом этапе клиенту главное найти и узнать, где находится сервер. Мы включили компьютер, который находится в сети (рис. 69). Еще в этой же сети работает DHCP. В данном случае наш компьютер — это клиент, а DHCP — сервер. Теперь нашему устройству необходимо получить IP-адрес и другую необходимую информацию о сети, например, шлюз, адреса DNS и маску подсети. Поэтому клиент начинает поиски сервера и посылает сообщение «DHCPDISCOVER» на компьютеры внутри этого сегмента сети.

Предложение (Offer): Сервер → Клиент

DHCP-сервер получает сообщение от клиента, после чего выбирает свободный IP-адрес из числа доступных и отправляет его в ответном сообщении «DHCPOFFER».

Как правило, IP-адрес закрепляется за клиентом на определенное время, поэтому может меняться между сеансами работы в сети.

Если клиенту ответили несколько серверов, он выберет какой-то один и получит от него IP-адрес с настройками.

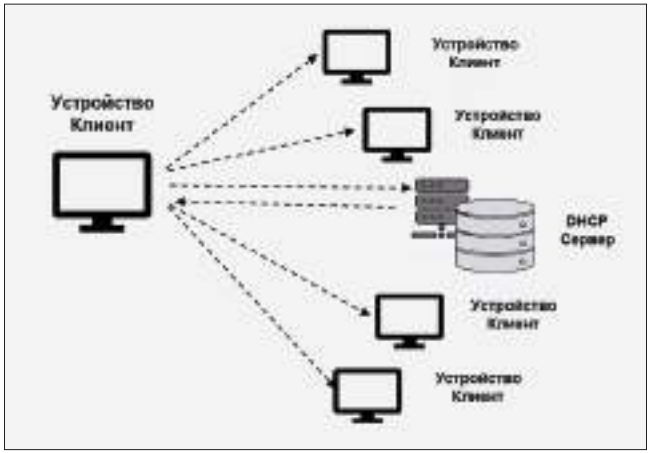

Рис. 69. Запрос клиента получат все участники сети, но ответит только сервер

Запрос (Request): Клиент → Сервер

Клиент получил IP-адрес и отправляет серверу ответное сообщение: «DHCPREQUEST». В нем он еще раз прописывает полученный адрес и тем самым подтверждает, что будет использовать его.

Ответное сообщение с IP-адресом получают все DHCP-серверы в сети, если их несколько. Это необходимо для того, чтобы каждый сервер знал, что этот адрес занят, и не предлагал его другим клиентам.

Только один сервер продолжает взаимодействие с клиентом. Это тот, который предложил выбранный клиентом IP-адрес.

Подтверждение (ACK): Сервер → Клиент

Сервер отправляет сообщение «DHCPACK» и тем самым закрепляет IP-адрес за клиентом. В сообщении содержится сам адрес, срок его использования и дополнительные настройки сети. Клиент проверяет эти настройки, применяет полученную конфигурацию и получает доступ к сети.

В общем виде весь процесс взаимодействия выглядит следующим образом.

Клиент: «Кто тут сервер? Мне надо получить IP-адрес: „DHCPDISCOVER”».

Сервер: «Я — сервер! Предлагаю тебе использовать вот этот IP: „DHCPOFFER"».

Клиент: «Хорошо, я буду использовать этот IP, что ты мне отправил: „DHCPREQUEST"».

Сервер: «Вот и договорились. Приятной работы в сети „DHCPACK"».

Другие варианты сообщений

«DHCPINFORM» — так клиент запрашивает локальные настройки сети. В ответ на это сообщение сервер посылает запрашиваемую конфигурацию.

«DHCPNAK» — так сервер отказывает клиенту пользоваться IP-адресом.

«DHCPRELEASE» — так клиент сообщает, что отключается от сети и освобождает свой IP-адрес. После этого сервер снова добавляет этот адрес в список доступных.

Перед тем как завершить работу и отключиться от сети, клиент автоматически отправит серверу сообщение «DHCPRELEASE». Это значит, что IP-адрес свободен и сервер может передать его другому компьютеру.

DHCP-сервер может назначать IP-адреса тремя способами:

— фиксированным (ручным);

— автоматически;

— динамически.

Фиксированный — в этом случае происходит настройка DHCP-сервера, в ходе которой администратор вручную прописывает соответствие между каждым MAC-адресом и IP-адресом. Таким образом, за каждым устройством закрепляется свой адрес, который будет выдавать сервер.

Это удобно в рамках небольшой сети, когда известны MAC-адреса всех компьютеров.

MAC-адрес, или физический адрес — это цифровой адрес устройства, который закреплен за сетевой картой «с завода». Благодаря ему провайдер знает, на какой компьютер направлять интернет-трафик.

Автоматический — при таком способе каждое устройство автоматически получает IP-адрес, который не будет меняться. DHCP-сервер выдает адрес в бессрочную аренду, пока клиент от него не откажется.

Динамический — DHCP-сервер выдает клиенту любой адрес из диапазона свободных. Эти адреса не закрепляются за конкретными устройствами.

Вспомним, что первое сообщение, которое отправляет клиент, чтобы найти сервер («DHCPDISCOVER»), — широковещательное. Следовательно, клиент и сервер из разных подсетей не смогут просто так взаимодействовать.

Эту проблему решают с помощью ретрансляции или DHCP relay. С помощью этой настройки маршрутизаторы смогут передавать только широковещательный трафик, который относится к протоколу DHCP.

Таким образом, если маршрутизатор может работать в режиме DHCP relay, то у него получится передать первый запрос клиента «DHCPDISCOVER» серверу в другой подсети. В этом случае взаимодействию «клиент — сервер» ничего не помешает.

Функции DHCP

1. Для связи с другими устройствами и работы в сети компьютеру требуется IP-адрес.

2. Устройство может получить статический или динамический IP-адрес.

3. Динамические IP-адреса назначаются с помощью протокола DHCP.

4. Чтобы получить IP-адрес, устройство-клиент взаимодействует с DHCP-сервером по модели «DORA».

5. Сервер может назначить IP-адрес клиенту тремя способами: фиксированным, автоматически или динамически.

6. Если сервер и клиент находятся в разных подсетях, они смогут взаимодействовать с помощью ретрансляции DHCP relay.

Основная функция протокола DHCP — предоставление в аренду IP-адреса. Однако для правильной работы в сети TCP/IP хосту необходим еще ряд параметров, которые также можно распространять посредством DHCP. Набор параметров указан в RFC 2132.

Перечислим только **основные параметры:**

— Subnet mask — маска подсети;

— Router — список IP-адресов маршрутизаторов;

— Domain Name Servers — список адресов DNS-серверов;

— DNS Domain Name — DNS-суффикс клиента;

— WINS Server Names — список адресов WINS-серверов;
— LeaseTime — срок аренды (в секундах);
— Renewal Time (T1) — период времени, через который клиент начинает продлевать аренду;
— Rebinding Time (T2) — период времени, через который клиент начинает осуществлять широковещательные запросы на продление аренды.

Параметры могут применяться на следующих уровнях:
— уровень сервера;
— уровень области действия;
— уровень класса;
— уровень клиента (для зарезервированных адресов).

При настройке областей действия перед администратором встает вопрос: какой диапазон адресов выбрать для сети своей организации? Ответ зависит от того, подключена ли сеть к Интернету. Если сеть имеет доступ в Интернет, диапазон адресов назначается провайдером (ISP — Internet Service Provider, поставщик интернет-услуг) таким образом, чтобы обеспечить уникальность адресов в Интернете. Чаще всего бывает так, что провайдер выделяет один или несколько адресов для прямого доступа в Интернет, и они присваиваются прокси-серверам, почтовым серверам и другим хостам, которые являются буферными узлами между сетью организации и Интернетом. Большинство остальных хостов получают доступ к интернет-трафику через эти буферные узлы. В этом случае диапазон внутренних адресов организации должен выбираться из множества частных адресов.

Частные адреса (Private addresses), описанные в RFC 1918, специально выделены для применения во внутренних сетях и не могут быть присвоены хостам в Интернете. Существует три диапазона частных адресов:
— ID подсети — 10.0.0.0;
— ID подсети — 172.16.0.0;
— ID подсети — 192.168.0.0.

Внутри этих диапазонов адресов можно организовывать любые возможные подсети. Если сеть не имеет доступа в Интернет, то теоретически можно выбрать любой диапазон IP-адресов, не учитывая наличия хостов с такими же адресами в Интернете.

Следует отметить, что помимо описанных частных адресов существует диапазон *автоматических частных адресов* APIPA (Automatic Private IP Address): ID подсети — 169.254.0.0, маска подсети — 255.255.0.0. Адрес из этого диапазона выбирается хостом TCP/IP случайно, если отсутствует статический IP-адрес, DHCP-сервер не отвечает и не указан альтернативный статический адрес. После выбора IP-адреса хост продолжает посылать запросы DHCP-серверу каждые пять минут.

DHCP-серверы используют для хранения и доступа к базе данных механизм ESE (Extensible Storage Engine). Он же используется для хранения и доступа к данным каталога Active Directory и базе данных Microsoft Exchange.

5.7. Система доменных имен

Домен — это онлайн-адрес сайта, место его размещения в Интернете. С технической позиции доменный адрес — запись в базе данных. Когда пользователь указывает в поисковой строке доменное имя, компьютер понимает, какой сайт необходимо показать и по какому адресу отправить запрос.

Символьный адрес (домен) не является обязательным, но он упрощает работу пользователей в сети. Существует два типа символьных имен, которые используются в IP-сетях:

— DNS-имя (RFC 1034, 1035);

— NetBIOS-имена.

Фрагмент системы доменных имен интернет-пространства представлен на рис. 70.

Корневой домен как реальный узел не существует — он исполняет роль вершины дерева. Его потомки (поддомены) — это домены 1-го (верхнего) уровня. Их можно условно разделить на три группы:

— name — домен, который используется для преобразования IP-адресов в доменное имя (обратное преобразование);

— домены организаций (com, org, net и т. д.);

— географические домены стран — имена для доменов, зарегистрированных в соответствующих странах (например, *ru* — для России, *ua* — для Украины, *uk* — для Великобритании и т. д.).

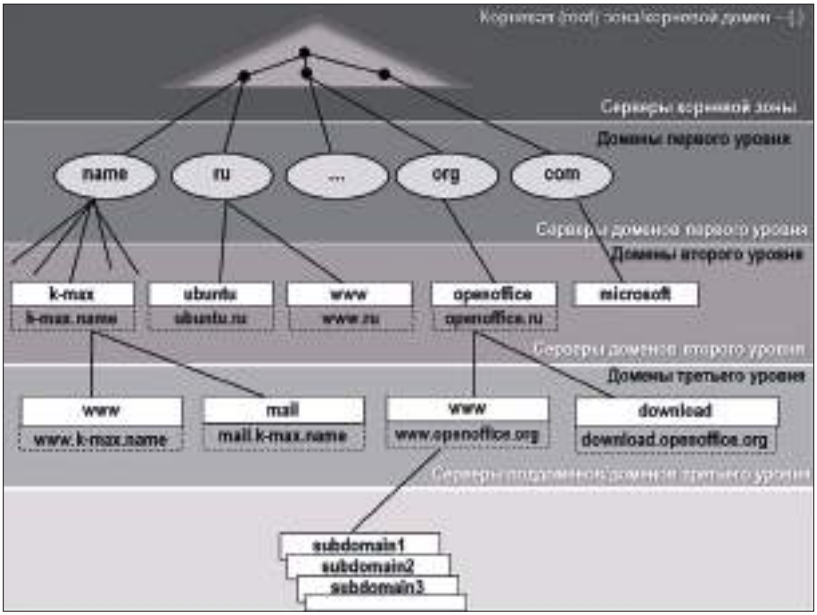

Рис. 70. Система доменных имен

Домен 1-го уровня включает в себя только домены 2-го. Записи об отдельных хостах могут содержаться в доменах, начиная со 2-го. Созданием и управлением домена 1-го уровня занимается международная организация ICANN. Домены 2-го уровня, находящиеся в географических доменах, распределяются специальными национальными организациями. Управлением доменами 3-го и следующего уровней занимаются владельцы соответствующих доменов 2-го уровня.

Полностью доменное имя FQDN записывается следующим образом: имя хоста (лист в дереве пространства имен), затем через точку следует DNS-суффикс, запись заканчивается точкой, после которой подразумевается корневой домен. Например, www.vshu.kirov.ru. При этом www — имя хоста, а vshu.kirov.ru — DNS-суффикс.

Для согласования двух систем адресаций необходима служба, которая занимается преобразованием доменных имен в IP-адреса и обратно. Данные функции выполняет служба

DNS. Процесс преобразования доменного имени в IP-адрес называется разрешением доменного имени. Простейшим способом разрешения доменного имени является файл hosts. Такой прием используется, как правило, в небольших сетях.

Служба поддерживает распределенную базу данных, которая хранится на специальных компьютерах (DNS-серверах). Вся информация не хранится в одном месте, ее части распределены по отдельным DNS-серверам. Так, например, за домены 1-го уровня отвечают 13 корневых серверов, имеющих имена от A.ROOT-SERVERS.NET до M.ROOT-SERVERS.NET, расположенных по всему миру. Такие части пространства называются зонами.

Для преобразования IP-адресов в DNS существуют зоны обратного преобразования (reverse lookup zone). На верхнем уровне пространства имен Интернета этим зонам соответствует домен in-addr.name.

Следуя правилам формирования DNS-имен, зона обратного преобразования, соответствующая подсети 156.98.10.0, будет называться 10.98.156.in-addr.name.

5.8. Процесс разрешения имен

В процессе разрешения участвуют DNS-клиент и DNS-сервер. Системный компонент DNS-клиента, называющийся DNS-распознавателем, отправляет запросы на DNS-серверы. Бывает двух видов:

— интерактивные — DNS-сервер обращается к DNS-серверу с просьбой разрешить имя без обращения к другим DNS-серверам;

— рекурсивные — всю работу по разрешению имени выполняет DNS-сервер путем отправки запросов другим DNS-серверам. DNS-сервер всегда сначала ищет имя в собственной базе данных или в кэше, в случае отсутствия обращается к другим серверам.

В основном DNS-клиентами используются рекурсивные запросы. На рис. 71 проиллюстрирован процесс разрешения доменного имени с помощью рекурсивного запроса.

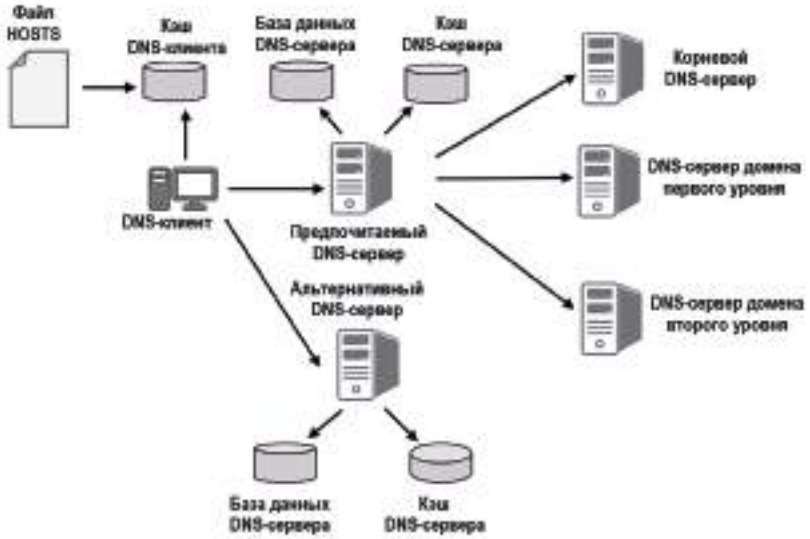

Рис. 71. Процесс рекурсивного разрешения имен

Сначала DNS-клиент осуществляет поиск в собственном локальном кэше DNS-имен. Это память для временного хранения ранее разрешенных запросов. В эту же память переносится содержимое файла HOSTS (каталог windows/system32/drivers/etc). Утилита IPconfig с ключом /displaydns отображает содержимое DNS-кэша. Если кэш не содержит требуемой информации, DNS-клиент обращается с рекурсивным запросом к предпочитаемому DNS-серверу (Preferred DNS server), адрес которого указывается при настройке стека TCP/IP. DNS-сервер просматривает собственную базу данных, а также кэш-память, в которой хранятся ответы на предыдущие запросы, отсутствующие в базе данных. В том случае, если запрашиваемое доменное имя не найдено, DNS-сервер осуществляет итеративные запросы к DNS-серверам верхних уровней, начиная с корневого DNS-сервера.

Просмотр DNS-кэша осуществляется утилитой ipconfig/displaydns. Очистка кэша — ipconfig / flushdns.

Выводы к главе 5

В пятой главе описаны основные цели сетевого администрирования, рассмотрены типовые задачи, с решением которых постоянно сталкиваются администраторы. Приведены основные методы решения представленных типовых задач.

Рассмотрены теоретические основы построения RAID-массивов, основные принципы, структура одиночных RAID-массивов, позволяющие понимать принципы построения составных, уметь их анализировать и сравнивать с точки зрения эффективности использования дискового пространства.

Представлены вопросы, связанные с решением задач сетевой адресации в распределенных информационных системах, описаны как правила использования IP-адресов, так и приведены примеры, затрагивающие определение типа адреса, идентификатора сети, идентификатора узла.

Рассмотрены вопросы, связанные с решением задач динамической адресации в информационных системах, описаны основные принципы и правила работы DHCP-сервера, подробно рассмотрен процесс получения клиентом IP-адреса.

Также рассмотрены вопросы, связанные с решением задач символьной адресации (DNS, NetBios) в информационных системах, описаны основные принципы и правила работы DNS-сервера и службы DNS, подробно рассмотрен процесс разрешения клиентом символьного адреса, рассмотрена структура базы данных DNS-сервера. Также приведена структура NetBios-имени, способы определения его типа, методы разрешения.

Вопросы для самоконтроля

1. Дайте определение информационной системы.
2. Основные цели и задачи сетевого администрирования.
3. Опишите модель межсетевого взаимодействия OSI.
4. Опишите модель межсетевого взаимодействия TCP/IP.
5. Какова основная цель сетевого администрирования?
6. Назовите основные инструменты администрирования.
7. Назовите основные виды задач сетевого администрирования.

8. Технологии, используемые при построении RAID-массивов.
9. RAID 0. Основные достоинства и недостатки.
10. RAID 1. Основные достоинства и недостатки.
11. RAID 2. Основные достоинства и недостатки.
12. RAID 3 и 4. Основные достоинства и недостатки.
13. RAID 5. Основные достоинства и недостатки.
14. Назначение IP-адреса.
15. Структура IPv4-адреса.
16. NETWORK ID и HOST ID в IPv4.
17. Использование масок для определения NETWORK и HOST ID.
18. Особенности IPv6-адресации.
19. Необходимые адреса, которые должны распознавать узлы.
20. Адреса, которые должны распознавать маршрутизаторы.
21. Адреса, которые должны распознавать приложения.
22. Какие вы знаете протоколы маршрутизации?
23. Статическая маршрутизация, ее достоинства и недостатки.
24. Авто-статическая маршрутизации, ее достоинства и недостатки.
25. Для решения какой проблемы предназначен протокол DHCP?
26. Что такое область действия?
27. Перечислите основные параметры DHCP.
28. Поясните значение сообщений DHCPDISCOVER, DHCPOFFER, DHCPREQUEST, DHCPACK.
29. По диаграмме переходов объясните принципы работы DHCP-клиента.
30. Для чего необходимы доменные имена?
31. Для чего нужна служба DNS?
32. Что такое корневой домен?
33. Каково было предназначение файла hosts?
34. Чем отличается служба DNS от системы имен DNS?
35. Принципы разрешения NetBios-имен.
36. Какие символы разрешены в DNS-именах?

ГЛАВА 6

Кибербезопасность информационных систем

Безопасность ИС — защищенность данных и инфраструктуры, которая их поддерживает, от ненамеренных и специальных искусственных и естественных воздействий, способных нарушить целостность, доступность, конфиденциальность сведений.

Определено, что ИС функционирует в киберпространстве, понимаемом как сфера деятельности в информационном пространстве, образованная совокупностью коммуникационных каналов Интернета и других телекоммуникационных сетей, технологической инфраструктурой, обеспечивающей их функционирование, и любых форм осуществляемой посредством их использования человеческой активности (личности, организации, государства).

Кибербезопасность — совокупность условий, при которых все составляющие киберпространства защищены от максимально возможного числа угроз и воздействий с нежелательными последствиями.

В нашей реальности для ИС существует множество угроз — может быть взломано наше устройство (компьютер) и могут быть зашифрованы все ваши файлы, могут украсть данные и использовать их в своих целях, могут выкрасть коммерческую информацию компании и продать ее конкурентам, а могут использовать устройство для кибератак на серверы крупных компаний или с целью шифровать собственный трафик.

Кибербезопасность — это набор инструментов, которые защищают нас от подобных угроз. Это может быть софт, например, антивирус, или устройство для фильтрации трафика вроде Firewall, которое шифрует данные или просто определяет набор рекомендаций, как вести себя, чтобы защитить данный трафик.

В учебном пособии рассмотрим основные составляющие кибербезопасности ИС.

6.1. Понятие Firewall

Информационные системы в компьютерной сети и сама компьютерная сеть изначально являются незащищенными и уязвимыми для внешних компьютерных атак.

Чтобы предотвратить несанкционированный доступ к устройству, подключенному к глобальной или частной сети, необходимо использовать специальные программные средства, называемые Firewall, «сетевой фильтр», «брандмауэр» (все названия являются синонимами).

Firewall переводится с английского как «огненная стена». Суть этой «стены» — защита компьютера, локальной сети или отдельных узлов от внешних атак, вредоносного программного обеспечения, фильтрация входящих и исходящих пакетов данных, обеспечение дополнительной безопасности при работе в сети.

Firewall подразделяются на разные типы, в зависимости от своих характеристик и выполняемых функций. Это могут быть:

— коммутаторы;

— фильтры сетевого уровня с анализом IP-адреса отправителя и получателя;

— шлюзы для контроля состояния канала сеансового уровня;

— шлюз прикладного уровня (прокси-сервер);

— брандмауэр с динамической фильтрацией входящих и исходящих пакетов.

Firewall можно устанавливать как на персональный компьютер (устройство) поверх операционной системы для защиты непосредственно этой машины, так и на сеть — для выполнения функции шлюза данной сети. Исходя из этого, брандмауэр может именоваться **host-based** или **network.**

Сетевой Firewall, основанный на стандартном ПК, называется *PC-based.* Если функционал Firewall разработан на аппаратном уровне отдельной системы, то это ASIC-**accelerated Firewall.**

Установка и настройка сетевого фильтра должна производиться администратором по сетевой безопасности, так как некомпетентное вмешательство в функционал брандмауэра может нанести существенный вред защищаемой сети (могут быть за-

прещены или ограничены в действиях некоторые необходимые службы).

К основным функциям Firewall, обеспечивающим защиту компьютера или сети от внешних угроз, относят:

— ограничение и контроль доступа к незащищенным службам узла сети;

— формирование регламента порядка доступа к службам;

— регистрирование и учет попыток доступа к устройству извне и от объектов внутренней сети;

— препятствование получению информации об устройстве или сети;

— трансляция ложных данных о защищаемой сети.

Использование Firewall, несомненно, приносит существенную пользу, но в то же время ощутимо увеличивает время отклика сети и снижает ее пропускную способность, так как на фильтрацию всех пакетов требуется определенное время.

Необходимо добавить, что сетевой фильтр не защитит компьютер от загружаемого непосредственно пользователем вредоносного программного контента (вирусов), а также от утечки персональных данных. Для этих целей рекомендуется использовать антивирусы и соблюдать условия конфиденциальности в сети.

6.2. Антивирусы

Антивирус — это компьютерная программа (средство антивирусной защиты), направленная на обнаружение компьютерных вирусов, а также нежелательных (считающихся вредоносными) программ и восстановление зараженных (модифицированных) такими программами файлов и профилактику — предотвращение заражения (модификации) файлов или операционной системы вредоносным кодом.

В нашей стране применяется один из лучших в мире антивирус Касперского[1].

Антивирус Касперского — антивирусное программное обеспечение, разработанное лабораторией Касперского, ко-

[1] URL: https://kaspersky.ru.

торое предоставляет пользовательским компьютерам защиту от различных вирусов, троянов, шпионских программ, руткитов и adware (реклама). Кроме того, антивирус Касперского предоставляет проактивную защиту с компонентом HIPS (Host Intrusion Prevention System) против пока еще неизвестных угроз.

Кроме антивирусной программы лабораторией Касперского выпускается бесплатная утилита для лечения компьютера Kaspersky Virus Removal Tool.

Лаборатория Касперского не просто предоставляет антивирусное программное обеспечение, но и обеспечивает круглосуточную техническую поддержку.

На Российском рынке антивирусных продуктов программа Касперского занимает долю более 50 %.

С 2002 г. Касперский выпускает межсетевой экран для защиты от хакеров — Kaspersky Anti-Hacker — и систему защиты от спама Kaspersky Anti-Spam.

Сегодня Лаборатория Касперского предоставляет покупателям целый ряд продуктов для борьбы с вирусами и вредоносными программами. Один из наиболее популярных на сегодняшний день продукт — это Kaspersky internet security. Данный продукт, кроме собственно антивирусного модуля, оснащен компонентом сетевого экрана, предназначенным для защиты пользовательского компьютера от хакерских атак.

Антивирус Касперского предоставляет несколько видов защиты компьютеров от вредоносных программ и вирусов: базовую защиту, проактивную защиту (превентивные меры для предотвращения угрозы), услуги по восстановлению системы и утраченных данных, а также защиту конфиденциальных данных.

В базовую защиту входят:

— защита от вирусов, червей и троянов;

— защита от рекламных и шпионских программ;

— проверка файлов (как в автоматическом режиме, так и по требованию);

— проверка входящих писем в почтовой программе (подходит для любых почтовых клиентов);

— проверка трафика;

— защита мессенджеров типа MSN или ICQ;

— проактивная защита от новых потенциально вредоносных программ;

— проверка Java-скриптов и Visual Basic-скриптов;

— защита от неявных «битых» ссылок;

— регулярная проверка файлов в автономном режиме;

— защита от фишинговых сайтов.

С целью предотвращения угрозы заражения Антивирус Касперского предлагает следующие услуги:

— поиск уязвимых мест в конкретной операционной системе и установленном программном обеспечении;

— обнаружение, анализ и устранение уязвимостей в браузере Internet Explorer;

— блокировка ссылок на зараженные и подозрительные сайты;

— распознавание вирусов по способу их упаковки;

— глобальный мониторинг угроз.

В случае, если система уже была поражена и требуется восстановление данных, антивирус Касперского предполагает возможность установки программ непосредственно на уже зараженный компьютер. Кроме того, работающая антивирусная программа защищена от остановки или выключения. После того как вредоносные программы будут удалены, включается функция восстановления корректных настроек в системе. Также антивирусник располагает инструментами для того, чтобы возможно было создавать диск аварийного восстановления.

Описание всех функций и технологий антивирусной программы Касперского представлено на официальном сайте.

Для ИС важную роль играет аутентификация и авторизация данных и пользователей, а также кодирование (шифрование) данных, или криптография.

6.3. Основы криптографии, алгоритм DES

Мы ежедневно сталкиваемся с криптографией в различных ИС— когда хотим что-то оплатить в интернете, авторизоваться на сайте или подписать электронные документы.

Криптографические методы помогают защитить персональные данные и обеспечивают безопасную передачу информации ИС в сети.

Криптографические методы защиты информации — это специальные методы шифрования, кодирования или иного преобразования информации, в результате которого ее содержание становится недоступным для посторонних лиц без предъявления ключа криптограммы и обратного преобразования (рис. 72).

Рис. 72. Методы криптографической защиты данных

Криптографические методы защиты информации — это специальные методы шифрования, кодирования или иного преобразования информации, в результате которого ее содержание становится недоступным для посторонних лиц без предъявления ключа криптограммы и обратного преобразования.

Для повышения эффективности криптографических методов при обработке и передаче данных средствами вычислительной техники был разработан стандарт шифрования данных Data Encryption Standard (DES). Международная организация стандартизации ISO (International Organization for Standardization) зарегистрировала алгоритм как **DEA-1**.

Основные свойства алгоритма DEA-1:

— используется только один ключ длиной 56 битов;

— зашифровав сообщение с помощью одного пакета, для расшифровки вы можете использовать любой другой;

— относительная простота алгоритма обеспечивает высокую скорость обработки информации;

— достаточно высокая стойкость алгоритма.

DES осуществляет шифрование 64-битовых блоков данных с помощью 56-битового ключа. Расшифрование в DES является операцией, обратной шифрованию, и выполняется путем повторения операций шифрования в обратной последовательности. Несмотря на кажущуюся очевидность, так делается далеко не всегда. Позже мы рассмотрим шифры, в которых шифрование и расшифрование осуществляются по разным алгоритмам.

Процесс шифрования заключается в начальной перестановке битов 64-битового блока, 16 циклах шифрования и, наконец, обратной перестановке битов (рис. 73).

Рис. 73. Обобщенная схема шифрования в алгоритме DES

Алгоритм DES представляет собой блочный шифр, он шифрует данные 64-битовыми блоками. На вход алгоритма поступает 64-битовый блок открытого текста, а на выходе получается 64-битовый блок шифротекста.

Алгоритм DES является симметричным: для шифрования и расшифровки используется один и тот же алгоритм и ключ (за исключением небольших различий в расписании смены ключа).

Длина ключа равна 56 битам. Ключ обычно представляется в виде 64-битового числа, но каждый восьмой бит используется для проверки четности и игнорируется. Биты четности являются наименьшими значащими битами байтов ключа. Ключ может быть любым 56-битовым числом, и его можно изменить в любой момент времени. Несколько чисел считаются слабыми ключами,

но их можно легко избежать. Стойкость полностью определяется ключом.

На простейшем уровне алгоритм представляет собой всего лишь комбинацию двух основных методов шифрования: смещения и диффузии.

Основным конструктивным элементом алгоритма DES является простая комбинация этих методов (подстановка с последующей перестановкой), зависящая от ключа. Такой блок называется **раундом** (round). Алгоритм DES состоит из 16 раундов: одна и та же комбинация методов применяется к открытому тексту 16 раз (рис. 74).

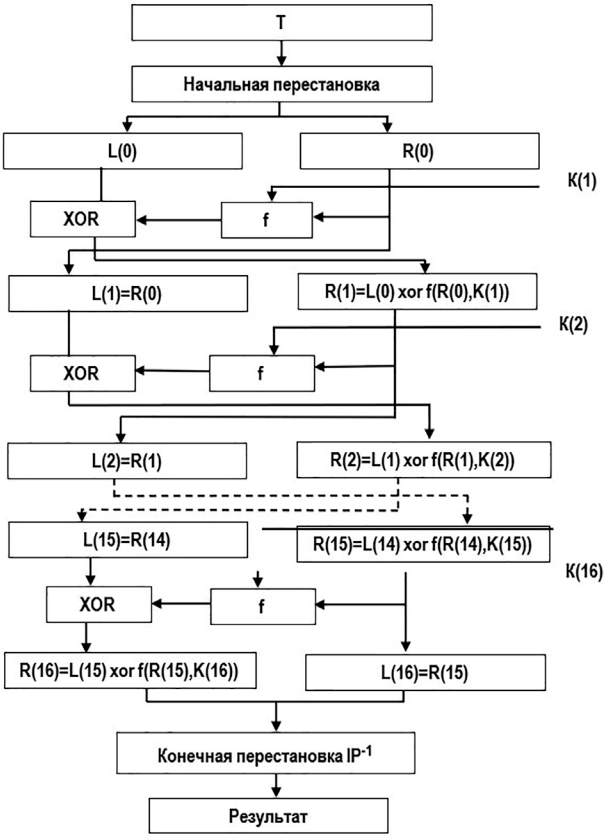

Рис. 74. Алгоритм DES

Алгоритм использует только стандартную арифметику и логические операции над 64-битовыми числами, поэтому он легко реализовывался в аппаратуре во второй половине 1970-х гг. Итеративная природа алгоритма делает его идеальным для реализации в специализированной микросхеме. Первоначальные программные реализации были довольно громоздкими, но современные программы намного лучше.

Схема алгоритма

Алгоритм DES работает с 64-битовым блоком открытого текста. После первоначальной перестановки блок разделяется на правую и левую половины, состоящие из 32 битов. Затем выполняется 16 раундов, включающих одинаковые операции, называемые функцией *f*, в которых данные объединяются с ключом. После 16-го раунда правая и левая половины объединяются, и в завершение выполняется заключительная перестановка (обратная по отношению к первоначальной).

В каждом раунде биты ключа сдвигаются, а затем из 56 битов ключа выбираются 48 битов (рис. 75). Правая половина данных увеличивается до 48 битов с помощью перестановки с расширением, складывается операцией XOR с 48 битами смещенного и переставленного ключа, проходит через 8 S-блоков, образуя 32 новых бита, и переставляется снова. Эти четыре операции выполняются функцией *f*. Затем результат функции складывается с левой половиной с помощью другой операции XOR. В итоге возникает новая правая половина, а старая правая половина становится новой левой. Эти операции повторяются 16 раз, образуя 16 раундов DES.

Если обозначить как B_i результат *i*-й итерации; L_i и R_i — левую и правую половины B_i; K_i — 48-битовый ключ для *i*-го раунда, а f — функцию, выполняющую все подстановки, перестановки и операцию XOR с ключом, то раунд можно представить как:

$$L_i = R_{i-1};$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

где: $\oplus$ — «исключающее или», т. е. $0 \oplus 0 = 0$; $0 \oplus 1 = 1$; $1 \oplus 0 = 1$; $1 \oplus 1 = 0$.

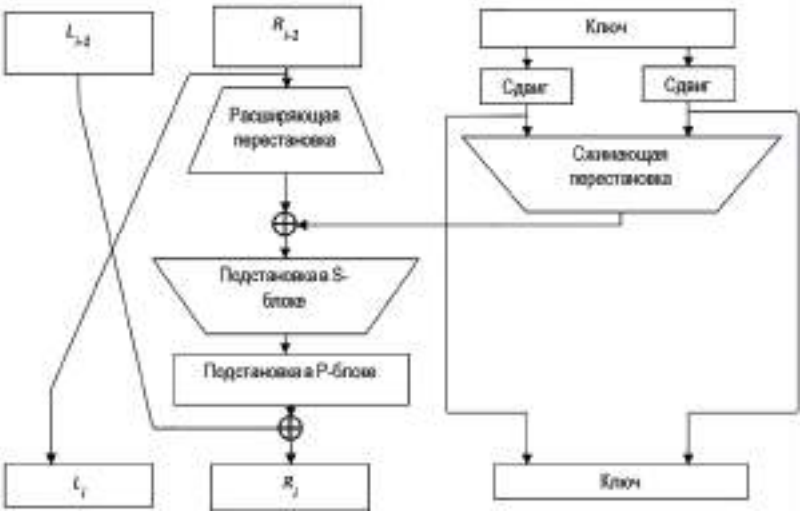

Рис. 75. Раунд алгоритма DES

Начальная перестановка

Начальная перестановка выполняется до первого раунда, при этом входной блок переставляется так, как показано в табл. 2. Эту и все остальные таблицы в этой главе следует читать слева направо и сверху вниз. Например, начальная перестановка перемещает бит 58 в битовую позицию 1, бит 50 — в битовую позицию 2, бит 42 — в битовую позицию 3 и т. д.

Таблица 2

Начальная перестановка

58	50	42	34	26	18	10	02
60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06
64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01
59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05
63	55	47	39	31	23	15	07

Начальная перестановка и соответствующая заключительная перестановка не влияют на стойкость алгоритма DES. Легко видеть, что основная цель этой перестановки — облегчение загрузки байтов открытого текста и шифротекста в микросхему DES. Напомним, что алгоритм DES появился до изобретения 16- и 32-битовых микропроцессорных шин. Поскольку программная реализация этой перестановки битов — трудная задача (в отличие от тривиальной реализации в аппаратном обеспечении), во многих программных реализациях алгоритма DES начальная и заключительная перестановки не используются. Такой новый алгоритм не менее стоек, чем DES, но он не соответствует стандарту DES, и поэтому не может так называться.

Преобразования ключа

Сначала 64-битовый ключ DES уменьшается до 56-битового ключа путем отбрасывания каждого восьмого бита, как показано в табл. 3. Эти биты используются только для контроля четности, чтобы проверять правильность ключа. После извлечения 56-битового ключа для каждого из 16 раундов DES генерируется новый 48-битовый **подключ** (subkey). Эти подключи K_i определяются следующим образом.

Таблица 3

Перестановка ключа

57,	49,	41,	33,	25,	17,	9,	1,	58,	50,	42,	34,	26,	18,
10,	2,	59,	51,	43,	35,	27,	19,	11,	3,	60,	52,	44,	36,
63,	55,	47,	39,	31,	23,	15,	7,	62,	54,	46,	38,	30,	22,
14,	6,	61,	53,	45,	37,	29,	21,	13,	5,	28,	20,	12,	4

Во-первых, 56-битовый ключ делится на две 28-битовых половины. Затем эти половины циклически сдвигаются налево на 1 или 2 бита в зависимости от раунда. Этот сдвиг показан в табл. 4.

Таблица 4

Количество сдвигаемых битов

Раунд	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Количество	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

После сдвига выбираются 48 из 56 битов. Поскольку при этом порядок следования битов в выбранном подмножестве изменяется, такая операция называется **сжимающей перестановкой** (compression permutation). Ее результатом является подмножество из 48 битов. Перестановка со сжатием (которая также называется **перестановочной выборкой**) определена в табл. 5. Например, бит сдвинутого ключа в позиции 33 перемещается в позицию 35-го результата, а 18-й бит сдвинутого ключа отбрасывается.

Таблица 5

Сжимающая перестановка

14,	17,	11,	24,	1,	5,	3,	28,	15,	6,	21,	10,
23,	19,	11,	4,	26,	8,	16,	7,	27,	20,	13,	2,
41,	52,	31,	37,	47,	55,	30,	40,	51,	45,	33,	48,
44,	49,	39,	56,	34,	53,	46,	42,	50,	36,	29,	32

Из-за сдвига в каждом из подключей используется другое подмножество битов. Каждый бит используется примерно в 14 из 16 подключей, хотя не все биты используются одинаковое количество раз.

Расширяющая перестановка

Эта операция расширяет правую половину данных R_i, с 32 до 48 битов. Поскольку при этом определенные биты не только повторяются, но и переставляются, эта операция называется **расширяющей перестановкой** (expansion permutation). Эта операция преследует две цели: выровнять размер правой половины в соответствии с ключом для выполнения операции XOR и получить более длинный результат, который можно будет сжать в ходе операции подстановки. Однако главная криптографическая цель этой операции заключается в том, что за счет влияния

одного бита на две подстановки зависимость битов результата от битов исходных данных возрастает быстрее. Это явление называется **лавинным эффектом** (avalanche effect). Алгоритм **DES** спроектирован так, чтобы зависимость каждого бита шифротекста от каждого бита открытого текста и каждого бита ключа возникала как можно быстрее.

Расширяющая перестановка показана на рис. 76. Иногда ее называют **E-блоком** (E-box)[2]. В каждом 4-битовом входном блоке первый и четвертый бит представляют собой 2 бита выходного блока, а второй и третий биты — 1 бит выходного блока. Соответствие между позициями результата и исходных данных показано в табл. 6. Например, бит входного блока в позиции 3 переместится в позицию 4 выходного блока, а бит входного блока в позиции 21 окажется в позиции 30 и 32 выходного блока.

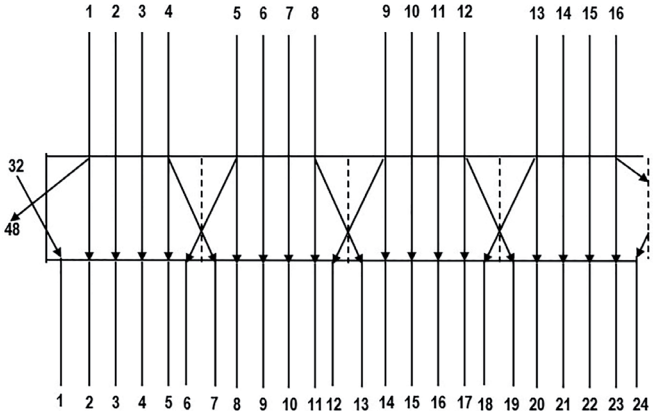

Рис. 76. Расширяющая перестановка

Таблица 6

Расширяющая перестановка

32,	1,	2,	3,	4,	5,	4,	5,	6,	7,	8,	9,
8,	9,	10,	11,	12,	13,	12,	13,	14,	15,	16,	17,
16,	17,	18,	19,	20,	21,	20,	19,	18,	17,	16,	15,
24,	25,	26,	27,	28,	29,	28,	29,	30,	31,	32,	33

Несмотря на то, что выходной блок больше входного, каждый входной блок порождает уникальный выходной блок.

Подстановка с помощью S-блоков

После применения операции XOR к сжатому и расширенному блокам к 48-битовому результату применяется операция подстановки. Подстановки производятся с помощью **8 блоков подстановки** (substitution boxes), или **S-блоков** (substitution). У каждого S-блока есть 6-битовый вход и 4-битовый выход. Всего существует 8 разных S-блоков. Для восьми S-блоков алгоритма DES требуется 256 байтов памяти. 48 битов делятся на 8 6-битовых подблоков. Каждый подблок обрабатывается отдельным S-блоком: первый подблок — первым S-блоком, второй — вторым S-блоком и т. д. (рис. 77).

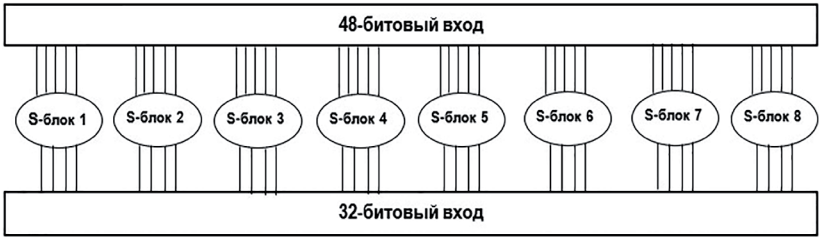

Рис. 77. Подстановка в S-блоках

Каждый S-блок представляет собой таблицу, состоящую из 2 строк и 16 столбцов. Каждый элемент в блоке является 4-битовым числом. 6 входных битов S-блока определяют, под какими номерами столбцов и строк следует искать выходное значение. Все 8 S-блоков показаны в табл. 7.

Входные биты определяют элемент S-блока особым образом. Рассмотрим 6-битовый вход S-блока: b_1, b_2, b_3, b_4, b_5 и b_6. Биты b_1 и b_6 объединяются, образуя 2-битовое число от 0 до 3, соответствующее строке таблицы. Средние биты b_2–b_5 объединяются, образуя 4-битовое число от 0 до 15, соответствующее столбцу таблицы.

Таблица 7

Подстановка в S-блоках

		Номер столбца																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Номер строки	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S1
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S2
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S3
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S4
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S5
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S6
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S7
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S8
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Например, пусть на вход шестого S-блока (т. е. биты функции XOR с 31 по 36) поступает число 110011. Первый и последний бит, объединяясь, образуют число 11, что соответствует строке 3 шестого S-блока. Средние 4 бита образуют число 1001, что соответствует столбцу 9 того же S-блока. Элемент шестого S-блока, находящийся на пересечении строки 3 и столбца 9, равен 14. Напомним, что строки и столбцы нумеруются с нуля, а не с единицы. Вместо числа 110011 подставляется число 1110.

Разумеется, намного проще программно реализовать S-блоки в виде массивов, содержащих 64 элемента. Для этого необходимо переупорядочить элементы, что совсем не сложно. Изменить индексы, не изменяя порядок следования элементов, недостаточно. S-блоки спроектированы очень тщательно. Однако такой способ описания S-блоков помогает продемонстрировать, как они работают. Каждый S-блок можно рассматривать как функцию подстановки 4-битового элемента: биты b_2–b_5 являются входом, а 4-битовое число — результатом. Биты b_1 и b_6 определяются соседними блоками и задают одну из четырех функций подстановки, возможных в данном S-блоке.

Подстановка с помощью S-блоков — ключевой этап алгоритма DES. Другие операции алгоритма являются линейными и легко поддаются анализу. S-блоки являются нелинейными, и именно они являются главным фактором, обеспечивающим стойкость алгоритма DES.

В результате раунда подстановки возникают 8 4-битовых блоков, которые вновь объединяются в единый 32-битовый блок. Этот блок поступает на вход следующего раунда — перестановки с помощью P-блоков.

Перестановка с помощью P-блоков

Результат подстановки с помощью S-блоков, содержащий 32 бита, перетасовывается с помощью **P-блока** (permutation). Эта перестановка перемещает каждый входной бит на другую позицию, ни один бит не используется дважды и ни один бит не игнорируется. Такая процедура называется **прямой перестановкой** (straight permutation), или просто **перестановкой.** Позиции, в которые перемещаются биты, показаны в табл. 8. Например, бит 21 перемещается в позицию 4, а бит 4 — в позицию 31.

Таблица 8

Перестановка с помощью P-блоков

16,	7,	20,	21,	29,	12,	28,	17,	1,	15,	23,	26,	5,	18,	31,	10,
2,	8,	24,	14,	32,	27,	3,	9,	19,	13,	30,	6,	22,	11,	4,	25

Наконец, к результату перестановки с помощью P-блока и левой половине первоначального 64-битового блока применяется операция XOR. После этого левая и правая половины меняются местами и начинается следующий раунд.

Заключительная перестановка

Заключительная перестановка является обратной по отношению к начальной. Она представлена в табл. 9. Обратите внимание, что после выполнения последнего раунда алгоритма DES левая и правая половины не меняются местами. Вместо этого объединенный блок $R_{16}L_{16}$ используется как вход заключительной перестановки. В этом нет ничего особенного; к такому же результату привела бы перестановка половин с последующим циклическим сдвигом. Благодаря этому алгоритм можно использовать как для шифрования, так и для расшифровки.

Таблица 9

Заключительная перестановка

40,	8,	48,	16,	56,	24,	64,	32,	39,	7,	47,	15,	55,	23,	63,	31,
38,	6,	46,	14,	54,	22,	62,	30,	37,	5,	45,	13,	53,	21,	61,	29,
36,	4,	44,	12,	52,	20,	60,	28,	35,	3,	43,	11,	51,	19,	59,	27,
34,	2,	42,	10,	50,	18,	58,	26,	33,	1,	41,	9,	49,	17,	57,	25

Расшифровка в алгоритме DES

После всех подстановок, перестановок, операций XOR и циклических сдвигов можно предположить, что алгоритм расшифровки совершенно отличается от алгоритма шифрования и является таким же сложным. Тем не менее, операции в алгоритме DES обладают очень полезным свойством: для шифрования и расшифровки применяется один и тот же алгоритм.

Алгоритм DES позволяет использовать одну и ту же функцию как для шифрования, так и для расшифровки блока. Единствен-

ное отличие состоит в том, что ключи должны использоваться в обратном порядке. Иначе говоря, если в каждом раунде шифрования использовались ключи K_1, K_2, …, K_{16}, то ключами расшифровки будут K_{16}, K_{15}, …, K_1. Алгоритм, генерирующий ключ каждого раунда, также является циклическим. Ключ сдвигается направо, а количество позиций сдвига равно: 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.

6.4. Криптография в базах данных ИС

Современная криптография позволяет наиболее эффективно решать многие практические задачи в области защиты информации ИС — в частности, задачу защиты баз данных, которая в силу требований действующего законодательства и в силу практической необходимости на сегодняшний день стоит практически перед каждой реально работающей ИС.

Криптографические методы защиты данных, хранимых и обрабатываемых как записи БД, можно концептуально разделить на два основных класса: методы защиты от несанкционированного просмотра данных при помощи их шифрования как в процессе хранения, так и в процессе передачи по каналам связи на компьютеры пользователей; методы авторизации доступа пользователей к различным разделам БД, а также аутентификации блоков данных (записей БД). Шифрование непосредственно записей БД (или даже отдельных их частей) на различных ключах при использовании современных стойких алгоритмов позволяет радикально решить проблему надежного разграничения доступа к различным разделам БД и не нуждается в больших затратах на физическую защиту непосредственно самих носителей данных.

Аутентификация отдельных разделов БД, отдельных записей или даже отдельных их наиболее важных частей с помощью технологии электронной подписи позволяет гарантировать защиту от несанкционированных изменений в базе данных с любой наперед заданной степенью детализации и документирования.

Следует отметить тот факт, что на сегодня ни один способ зашиты баз данных, кроме криптографического, не может обеспечить того уровня надежности защиты данных и удобства

пользователей, которые обеспечиваются криптографическими технологиями.

Сквозная технология цифровой экономики определяет блокчейн. Блокчейн — это особый тип базы данных. Она отличается от типичной базы данных способом хранения информации; блокчейны хранят данные в блоках, которые затем объединяются в цепочку. Блоки включают хеш-значение предыдущего блока, чтобы криптографически связать их вместе в цепочку, сохраняя хронологический порядок реестра.

Отметим еще одно обстоятельство — электронная подпись в БД. Основа электронной подписи — криптография.

Будем в дальнейшем использовать терминологию криптографии.

Любой открытый (исходный) текст образует сообщение. Процесс маскировки сообщения, скрывающий его содержание, называется шифрованием. Зашифрованное сообщение называется шифротекстом. Процесс преобразования шифротекста обратно в открытый текст называется расшифровкой. Эта последовательность действий показана на рис. 78.

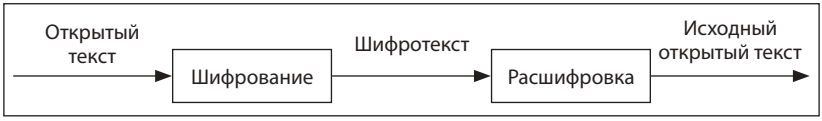

Рис. 78. Шифрование и расшифровка

Обозначим открытый текст буквой *M* или *P*. Он может быть потоком битов, текстовым файлом, битовым изображением, оцифрованным звуком, цифровым видеоизображением... Чем угодно. В компьютере сообщение *M* представляет собой просто двоичные данные.

Открытый текст может быть предназначен для хранения или передачи. В любом случае, *M* — это сообщение, которое должно быть зашифровано.

Обозначим шифротекст буквой *C*. Он тоже представляет собой двоичные данные: иногда того же размера, что и *M*, иногда большего. Если шифрование выполняется одновременно со сжатием, шифротекст *C* может быть меньше, чем сообщение

M. Однако само по себе шифрование не подразумевает сжатия информации. Функция шифрования E действует на сообщение *M*, создавая шифротекст *C*. В математической форме это записывается так:

$$E(M) = C.$$

В обратном процессе функция расшифровки *D* действует на шифротекст *C*, восстанавливая сообщение *M*:

$$D(C) = M.$$

Поскольку цель шифрования и последующей расшифровки сообщения заключается в восстановлении исходного открытого текста, должно выполняться равенство:

$$D(E(M)) = M.$$

Кроме обеспечения конфиденциальности криптография применяется для решения следующих задач:

— **аутентификация.** Получатель сообщения должен иметь возможность установить его источник, злоумышленник не должен иметь возможности маскироваться под кого-то другого;

— **целостность.** Получатель сообщения может проверить, не было ли сообщение изменено в процессе доставки, и злоумышленник не подменял правильное сообщение ложным;

— **невозможность отказа от авторства.** Отправитель не должен иметь возможности ложно отрицать отправку сообщения.

Криптографический алгоритм, который также называется шифром, — это математическая функция, используемая для шифрования и расшифровки. Обычно эти функции связаны друг с другом: одна предназначена для шифрования, другая — для расшифровки.

Вводится понятие ключа, который обозначается буквой *K*. Ключ может быть любым значением, выбранным из большого множества. Множество возможных значений ключа называют пространством ключей. Этот ключ используется и при шифровании, и при расшифровке (т. е. они зависят от ключа, и этот факт обозначается индексом *K*). Итак, эти функции выглядят следующим образом:

$$E_K(M) = C;$$

$$D_K(C) = M.$$

При этом выполняется следующее равенство (рис. 79):

$$D_K(E_K(M)) = M.$$

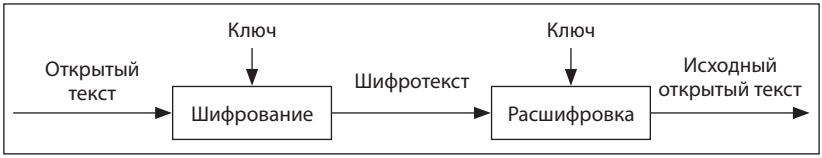

Рис. 79. Шифрование и расшифровка с ключом

В некоторых алгоритмах при шифровании и расшифровке используются разные ключи (рис. 80). Иначе говоря, ключ шифрования отличается от соответствующего ключа расшифровки K_2. В этом случае выполняются такие соотношения:

$$E_{K_1}(M) = C;$$
$$D_{K_2}(C) = M;$$
$$D_{K_2}(E_{K_1}(M)) = M.$$

Рис. 80. Шифрование и расшифровка с двумя разными ключами

Безопасность этих алгоритмов полностью основана на ключе (или ключах), а не на их деталях. Это означает, что алгоритм можно опубликовать и проанализировать. Программные продукты, использующие этот алгоритм, могут становиться массовыми. Не имеет значения, что злоумышленник знает ваш алгоритм: если ему не известен конкретный ключ, то он не может читать ваши сообщения.

Криптосистема — это алгоритм и все возможные открытые тексты, шифротексты и ключи.

Симметричные алгоритмы

Существуют два основных типа алгоритмов, основанных на использовании ключей: симметричные алгоритмы и алгоритмы с открытым ключом.

Симметричные алгоритмы иногда называют условными. Это алгоритмы, в которых ключ шифрования может быть вычислен по ключу расшифровки, и наоборот.

В большинстве симметричных алгоритмов ключи шифрования и расшифровки совпадают.

Эти алгоритмы, также называемые **алгоритмами с секретным ключом** или **алгоритмами с единственным ключом,** требуют, чтобы отправитель и получатель согласовали ключ до начала передачи секретных сообщений. Безопасность симметричного алгоритма зависит от ключа. Его раскрытие означает, что кто угодно сможет шифровать и расшифровывать сообщения. Пока передаваемые сообщения должны быть тайными, ключ должен храниться в секрете. Шифрование и расшифровка с использованием симметричного алгоритма записывается так:

$$E_K(M) = C;$$

$$D_K(C) = M.$$

Симметричные алгоритмы подразделяются на две категории. Одни алгоритмы обрабатывают открытый текст побитово (иногда побайтово). Они называются **потоковыми алгоритмами** или потоковыми шифрами. Другие обрабатывают биты открытого текста по группам. Группы битов называются **блоками,** а алгоритмы — **блочными алгоритмами** или **блочными шифрами.**

В современных компьютерных алгоритмах типичный размер блока составляет 64 бита — достаточно большое значение, чтобы предотвратить анализ, и в то же время достаточно небольшое для удобной работы.

Алгоритмы с открытым ключом

Алгоритмы с открытым ключом, называемые также асимметричными алгоритмами, устроены так, что ключ шифрования отличается от ключа расшифровки. Более того, ключ расшифров-

ки не может быть (по крайней мере, в течение разумного интервала времени) вычислен по ключу шифрования. Эти алгоритмы называют алгоритмами с открытым ключом, поскольку ключ шифрования может быть открыт.

Кто угодно может использовать этот ключ для шифрования сообщения, но только конкретный человек с соответствующим ключом расшифровки может расшифровать сообщение. В этих системах ключ шифрования часто называют открытым, а ключ расшифровки — закрытым.

Закрытый ключ иногда называется секретным ключом, но, чтобы не возникла путаница с симметричными алгоритмами, этот термин в учебном пособии не используется.

Шифрование с открытым ключом *K* обозначается следующим образом:

$$E_K(M) = C.$$

Несмотря на то, что открытый и закрытый ключи отличаются друг от друга, расшифровка с соответствующим закрытым ключом обозначается следующим образом:

$$D_K(C) = M.$$

Иногда сообщения шифруются закрытым ключом, а расшифровываются — открытым. Такой метод используется для цифровой подписи. Эти операции обозначаются следующим образом:

$$E_K(M) = C;$$

$$D_K(C) = M.$$

Криптографические алгоритмы либо заменяли одни символы другими, либо переставляли символы. Алгоритмы получили название `подстановочных и перестановочных шифров.

Подстановочным называется шифр, который каждый символ открытого текста заменяет другим символом в шифротексте. Получатель выполняет обратную подстановку в шифротексте, восстанавливая открытый текст. В классической криптографии существуют четыре типа подстановочных шифров:

— простой подстановочный шифр, или моноалфавитный шифр — это шифр, который заменяет каждый символ открыто-

го текста соответствующим символом шифротекста. Примером простых подстановочных шифров являются криптограммы в газетах;

— омофонический подстановочный шифр похож на простую подстановочную криптосистему, за исключением того, что один символ открытого текста заменяется несколькими символами шифротекста. Например, букве А может соответствовать набор чисел 5, 13, 25 или 56, букве В — 7, 19, 31 или 42 и т. д.;

— полиграммный подстановочный шифр — это шифр, который заменяет одни блоки символов другими. Например, символам ABC могут соответствовать символы LIO, символам JBB — символы LLL и т. д.;

— полиалфавитный подстановочный шифр состоит из нескольких простых подстановочных шифров. Например, можно использовать пять разных простых подстановочных фильтров так, что каждый символ открытого текста заменяется с использованием одного конкретного шифра.

В настоящее время в языке программирования Python разработаны и доступны алгоритмы и программы для всех видов и типов кодирования.

Рассмотрим знаменитый шифр Цезаря в языке программирования Python.

Шифр Цезаря, также известный как шифр сдвига, код Цезаря или сдвиг Цезаря — один из самых простых и наиболее широко известных методов шифрования. Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите (рис. 81). Например, в шифре со сдвигом вправо на 1, А была бы заменена на Б, Б станет Г и т. д.

Шифр Цезаря представляет собой простой подстановочный фильтр. На самом деле этот алгоритм еще проще, чем подстановочный, потому что алфавит шифротекста представляет собой результат смещения алфавита открытого текста, а не его случайную перестановку.

Рассмотрим один из самых эффективных алгоритмов кодирования — одноразовый блокнот.

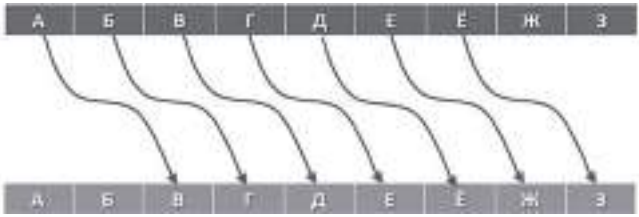

Рис. 81. Шифр Цезаря

Одноразовые блокноты, как ни странно, представляют близкую к идеальной схему шифрования.

Классический одноразовый блокнот представляет собой длинную неповторяющуюся последовательность совершенно случайных символов ключа, написанных на листах бумаги, склеенных в виде блокнота. В исходном варианте (придуманном в 1917 г.) в качестве блокнота использовалась одноразовая телетайпная лента.

Отправитель шифровал каждым символом ключа блокнота только один символ открытого текста. Такое шифрование представляет собой сложение по модулю 33 символов (алфавит) открытого текста и символа ключа из одноразового блокнота.

Каждый символ ключа используется только один раз и только в одном сообщении.

После шифрования сообщения отправитель уничтожает использованные страницы блокнота или использованную часть ленты.

Получатель расшифровывает каждый символ шифротекста, используя точно такой же блокнот и содержащиеся в нем ключи.

Расшифровав сообщение, получатель уничтожает те же самые страницы блокнота или фрагмент ленты.

В новом сообщении будут использованы новые символы ключа.

Идею одноразового блокнота можно легко применить к двоичным данным. Для этого используется одноразовый блокнот, содержащий не буквы, а биты.

Вместо суммирования открытого текста с ключом одноразового блокнота используется операция XOR.

Для расшифровки операция XOR применяется к шифротексту с тем же одноразовым блокнотом. Все остальное не меняется, и стойкость остается абсолютной.

Одноразовые блокноты используются и в настоящее время, главным образом, в сверхсекретных каналах связи с низкой пропускной способностью.

Собственноручные подписи издавна используются как доказательство авторства или, по крайней мере, согласия с документом. Подписавший не сможет впоследствии утверждать, что он не подписывал документ. Хотелось бы нечто подобное сделать на компьютерах, но возникает ряд проблем.

Во-первых, компьютерные файлы очень легко скопировать. Даже если собственноручную подпись человека трудно подделать (например, воспроизвести ее графическое изображение), то можно легко вырезать подлинную подпись из одного документа и вставить в другой.

Во-вторых, компьютерные файлы очень легко можно изменить после того, как они были подписаны, не оставляя никаких свидетельств изменения.

Для защиты подписи в компьютерном варианте было разработано множество алгоритмов цифровой подписи.

Все они представляют собой алгоритмы с открытым ключом, которые используют секретную информацию для подписи документов и открытую информацию для верификации подписи.

Иногда процесс подписания называют шифрованием с закрытым ключом, а процесс верификации подписи — расшифровкой с открытым ключом. Но это относится только к одному алгоритму — RSA (фамилии авторов — Rivest, Shamir и Adleman).

Другие алгоритмы реализуются иначе. Например, использование односторонних хеш-функций и меток времени добавляет дополнительные этапы при подписании и верификации подписи.

Выводы к главе 6

Мы определили, что ИС функционирует в киберпространстве, понимаемое как сфера деятельности в информационном пространстве, и сформулировали кибербезопасность как совокупность условий, при которых все составляющие киберпро-

странства защищены от максимально возможного числа угроз и воздействий с нежелательными последствиями.

В данной главе были рассмотрены основные составляющие кибербезопасности ИС.

Для предотвращения несанкционированного доступа к устройству ИС, подключенному к глобальной сети Интернет, необходимо использовать специальные программные средства, называемые Firewall, «сетевой фильтр», «брандмауэр» (все названия являются синонимами).

Для обнаружения компьютерных вирусов определен самый эффективный антивирус — программы Касперского.

Рассмотрены криптографические методы защиты БД ИС, которые делают недоступным содержание БД для посторонних лиц без предъявления ключа криптограммы и обратного преобразования.

Изучен стандарт шифрования данных Data Encryption Standard (DES), который Международной организацией стандартизации ISO (International Organization for Standardization) зарегистрирован как алгоритм как DEA-1.

Криптографические методы защиты данных, хранимых и обрабатываемых как записи БД, можно отнести к одному из двух основных классов:

— методы защиты от несанкционированного просмотра данных при помощи их шифрования как в процессе хранения, так и в процессе передачи по каналам связи на компьютеры пользователей;

— методы авторизации доступа пользователей к различным разделам БД, а также аутентификации блоков данных (записей БД).

В настоящее время появился такой новый тип базы данных, как сквозная технология цифровой экономики РФ. Показано, что основой надежного функционирования блокчейн является криптография.

Вопросы для самоконтроля

1. Охарактеризуйте место и роль ИС в управлении организацией.

2. Какие факторы определяют актуальность проблемы защиты ИС?

3. Перечислите особенности современных ИС как объектов защиты.

4. Дайте определение понятий «угроза», «уязвимость» и «атака».

5. Безопасность ИС.

6. Угрозы ИС.

7. Что такое среда Firewall?

8. Функции Firewall, обеспечивающие защиту компьютера.

9. Назначение и функции программы типа «Антивирус».

10. Предложения лаборатории Касперского.

11. Криптографические методы как средства защиты ИС.

12. Методы криптографической защиты данных.

13. Стандарт шифрования данных Data Encryption Standard (DES).

14. Схема алгоритма Алгоритм DES.

15. Раунд алгоритма DES.

16. Расшифровка в алгоритме DES.

17. Криптография в базах данных ИС.

18. Аутентификация отдельных разделов БД.

19. Блокчейн — особый тип базы данных.

20. Шифрование и расшифровка с ключом.

21. Шифрование и расшифровка с двумя разными ключами.

22. Алгоритмы с открытым ключом и технология блокчейн.

23. Шифр Цезаря.

24. Алгоритмов кодирования — одноразовый блокнот.

25. Что такое электронная подпись?

ЗАКЛЮЧЕНИЕ

Цифровая экономика РФ, сквозные технологии и, прежде всего, Big Data, блокчейн определили беспрецедентные преобразования в индустрии БД.

Жизненные циклы внедрения технологий БД ускорились. Архитектура БД развивается столь быстро, что привычные задачи уже не требуют решения, а связанные с ними навыки утратили актуальность. Изменились методы и способы администрирования и кибербезопасности.

В учебном пособии мы рассмотрели основные темы из области администрирования ИС, предполагающие, что администрирование ИС заключается в предоставлении пользователям соответствующих прав использования возможностей работы с системой (базой данных, нейронной сетью), в обеспечении целостности данных, а также в создании многопользовательских приложений. Администрирование ИС — это ее установка, управление доступом к ней, обеспечение целостности ИС и др.

В учебном пособии вопросы администрирования ИС мы рассматривали с использованием MS SQL-2019 и SQL СУБД Greenplum, популярной для Big Data.

Мы изучили общие систематизированные сведения об организации и структуре ИС. В доступной форме даны базовые понятия, цели, задачи администрирования ИС, принципы и правила настройки и использования стека протоколов TCP/IP, методы настройки различных типов адресации в IP-сетях. Рассмотрели эффективные решения задач управления пользователями и ресурсами сети, а также приобретение необходимых знаний и навыков в области кибербезопасности функционирования ИС.

В заключение необходимо отметить, что все рассмотренные модели и средства администрирования и кибербезопасности имеют право на существование и могут достаточно эффективно использоваться в конкретных случаях. Спектр технологий обработки больших данных велик, расширен новой технологией

блокчейн, и выбор той или иной технологии также обусловлен спецификой задачи, которую требуется решить в конкретных условиях, в конкретное время и обладая определенными ограниченными ресурсами.

Технология больших данных — блокчейн — несомненно, и дальше будет развиваться быстрыми темпами и доминировать в структуре ИС.

Для дальнейшего освоения дисциплины авторы рекомендуют изучить основы аналитики больших данных в среде машинного обучения SQL Server 2019 с использованием библиотек языка Python и внедрение методов кибербезопасности с использованием разработок лаборатории Касперского.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Бен-Ган И., Сарка Д., Талмейдж Р. Microsoft SQL Server 2012. Создание запросов. Учебный курс Microsoft: пер. с англ. — М.: Русская редакция, 2014. — 720 с.

Бен-Ган И. Microsoft SQL Server 2012. Основы T-SQL / пер. с англ. М. А. Райтмана. — М.: Эксмо, 2015. — 400 с.

Бондарев В. В. Введение в информационную безопасность автоматизированных систем: учеб. пособие. — 2-е изд. — М.: Изд-во МГТУ им. Н. Э. Баумана, 2018. — 250 с.

Диогенес Ю., Озкайя Э. Кибербезопасность: стратегии атак и обороны / пер. с англ. Д. А. Беликова. — М.: ДМКПресс, 2020. — 326 с.

Кэмпбелл Л., Мейджорс Ч. Базы данных. Инжиниринг надежности. — СПб.: Питер, 2020. — 304 с.

Левицкий Н. Д. Справочник системного администратора. Полное руководство по управлению Windows-сетью. — СПб.: Наука и техника, 2020. — 464 с.

Ложе И. Информационные системы: методы и средства. М.: Мир, 1979. — 632 с.

Microsoft Windows Server 2012. Полное руководство / Р. Моримото, М. Ноэл, Г. Ярдени и др.: — М.: Вильямс, 2013. — 1456 с.

Омассон Ж.-Ф. О криптографии всерьез / пер. с англ. А. А. Слинкина. — М.: ДМК Пресс, 2021. — 328 с.

Свейгарт Э. Криптография и взлом шифров на Python: пер. с англ.— СПб.: Диалекти, 2020. — 512 с.

Силен Д., Мейсман А., Али М. Основы Data Science и Big Data. Python и наука о данных. — СПб.: Питер, 2018. — 336 с.

Станек У. Internet Information Services (1 IS) 7.0. Справочник администратора: пер. с англ. — М.: Русская Редакция; СПб.: БХВ-Петербург, 2013. — 528 с.

Сухомлин В. А. Введение в анализ информационных технологий: учебник. — М.: Горячая линия-Телеком, 2003. — 427 с.

Уорд Б. Инновации SQL Server 2019. Использование технологий больших данных и машинного обучения / пер. с англ. Н. Б. Желновой. — М.: ДМК Пресс, 2020. — 408 с.

Фримен А. ASP.NET Core MVC 2 с примерами на C# для профессионалов. — 7-е изд.: пер. с англ. — СПб.: Диалектика, 2019. — 1008 с.

Черчхауз Р. Коды и шифры. Юлий Цезарь, «Энигма» и Интернет: пер. с англ. — М.: Весь Мир, 2005. —320 с.

Шнайер Б. Прикладная криптография: протоколы, алгоритмы и исходный код на С. — 2-е юбил. изд.: пер. с англ. — СПб.: Альфа-книга, 2017. — 1040 с.

Freeman A. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor. London, UK, 2022. — 1252 p.

Информация об авторах

Часовских Виктор Петрович — профессор кафедры шахматного искусства и компьютерной математики УрГЭУ, доктор технических наук, профессор
e-mail: u2007u@ya.ru

Лабунец Валерий Григорьевич — профессор кафедры шахматного искусства и компьютерной математики УрГЭУ, доктор технических наук, профессор
e-mail: vlabunets05@yahoo.com

Стариков Евгений Николаевич — заведующий кафедрой шахматного искусства и компьютерной математики УрГЭУ, кандидат экономических наук, доцент
e-mail: starikov_en@usue.ru

Акчурина Галия Абдулазисовна — старший преподаватель кафедры шахматного искусства и компьютерной математики УрГЭУ
e-mail: gaa20111@yandex.ru

Кох Елена Викторовна — доцент кафедры шахматного искусства и компьютерной математики УрГЭУ, кандидат сельскохозяйственных наук, доцент
e-mail: elenakox@mail.ru

Учебное издание

Часовских Виктор Петрович,
Акчурина Галия Абдулазисовна,
Лабунец Валерий Григорьевич
и др.

Администрирование и кибербезопасность информационных систем

Учебное пособие

Корректор
В. К. Матвеев

Компьютерная верстка
Н. В. Троицкой

Поз. 78. Подписано в печать 28.12.2022.
Формат 60 × 84 1/16. Гарнитура Minion. Бумага офсетная. Печать плоская.
Уч.-изд. л. 6,0. Усл. печ. л. 10,2. Печ. л. 11,0. Заказ 500. Тираж 24 экз.
Издательство Уральского государственного экономического университета
620144, г. Екатеринбург, ул. 8 Марта/Народной Воли, 62/45

Отпечатано с готового оригинал-макета в подразделении оперативной полиграфии
Уральского государственного экономического университета

УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ
ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ