Programming the Perl DBI

Alligator Descartes, Tim Bunce

O'REILLY®

Программирование на Perl DBI

Аллигатор Декарт и Тим Банс

Символ

Санкт-Петербург
2001

Аллигатор Декарт, Тим Банс

Программирование на Perl DBI

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*В. Рижий*
Редакторы	*Е. Изотова, Н. Макарова*
Корректура	*С. Беляева*
Верстка	*Е. Изотова, Н. Макарова*

Декарт А., Банс Т.

Программирование на Perl DBI. – Пер. с англ. – СПб: Символ-Плюс, 2001. – 400 с., ил.

ISBN 5-93286-013-8

Данная книга будет полезна как новичкам, которые найдут в ней описание архитектуры DBI и подробные инструкции по написанию программ с помощью DBI, так и знатокам DBI, которым предназначено описание тонкостей использования DBI и специфических особенностей отдельных драйверов DBD.

DBI является основным интерфейсом программирования баз данных на Perl. Любая программа, использующая DBI, может работать с любой базой данных или даже одновременно с несколькими базами данных различных фирм, такими как Oracle, Sybase, Ingres, Informix, MySQL, Access и другие.

Издание содержит полный справочник по DBI. Книга написана с учетом того, что читатель имеет базовые навыки программирования на Perl и может писать простые сценарии.

ISBN 5-93286-013-8
ISBN 1-56592-699-4 (англ)

Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 567-8775. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 31.10.2000. Формат 70x100$^{1}/_{16}$. Бумага офсетная.
Печать офсетная. Объем 25 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с диапозитивов в ГПП «Печатный Двор» Министерства РФ
по делам печати, телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

Оглавление

Предисловие

DBI является стандартным интерфейсом баз данных для языка программирования Perl. DBI не зависит от базы данных, т. е. позволяет работать практически с любой базой данных, в том числе с Oracle, Sybase, Informix, Access, MySQL и т. д.

Мы предполагаем у читателей данной книги наличие некоторого опыта работы с Perl, но не ожидаем хорошего знакомства с собственно базами данных. Изложение в книге развертывается неспешно, описывая различные типы баз данных и знакомя читателя с общепринятой терминологией.

Эта книга посвящена не только DBI, она касается более общей темы хранения и извлечения данных из баз различного вида. Это выражается и в том, что книга состоит из двух взаимосвязанных, но независимых частей. В первой части рассказывается о способах хранения и извлечения данных без использования DBI, в то время как вторая, и значительно большая по размеру часть охватывает использование DBI и родственных технологий.

На протяжении всей книги предполагается, что у читателя есть базовые навыки программирования на Perl, и он может самостоятельно писать простые сценарии. Если ваш уровень знания Perl не достаточен, советуем прочесть какие-нибудь книги по Perl, перечисленные в разделе «Дополнительные источники информации» данного предисловия.

Если вы достаточно подготовлены, можете читать книгу не подряд, а в зависимости от того, какие темы вас больше интересуют. Если вас интересует только DBI, можете безболезненно пропустить главу 2. С другой стороны, если вы хорошо разбираетесь в SQL, то, вероятно, пропустите главу 3, которая огорчила бы вас отсутствием описания многих важных тонкостей. В главе 7 сравниваются DBI и ODBC, и она интересна, в основном, «пахарям» от баз данных, ревнителям проектирования и тем, кто отчаянно пытается перенести свои приложения `Win32::ODBC` на DBI.

Вот перечень содержания книги по главам:

Глава 1 «Введение».

Позволяет получить общее представление о книге.

Глава 2 «Основные базы данных, не использующие DBI».

В этой главе рассказывается об основах хранения и извлечения данных с помощью базовых функций Perl посредством одноуровневых баз данных, иначе называемых базами данных на плоских файлах (flat-file database), использующих разделители или поля фиксированной ширины, либо через модули, не использующие DBI, такие как `AnyDBM_File`, `Storable`, `Data::Dumper` и другие. Хотя в этой главе не упоминается непосредственно DBI, способ, которым модули `Storable` и `Data::Dumper` упаковывают структуры данных Perl в строки, может легко быть применен и к DBI.

Глава 3 «SQL и реляционные базы данных».

В этой главе дается обзор основ SQL и реляционных баз данных, а также объясняется, как можно писать простые, но мощные SQL-выражения для построения запросов к базам данных и обработки их результатов. Если вы в какой-то мере знакомы с SQL, можете пропустить эту главу. Если же вы не знаете SQL, советуем прочесть ее, поскольку в последующих главах предполагается наличие базовых знаний об SQL и реляционных базах данных.

Глава 4 «Программирование с помощью DBI».

Эта глава знакомит с DBI путем изложения его архитектуры и основных DBI-операций, таких как соединение с базами данных и обработка ошибок. Эту главу важно прочесть, поскольку в ней описывается структура, которую предоставляет DBI для написания простых, мощных и надежных программ.

Глава 5 «Взаимодействие с базами данных».

В этой главе изложен основной материал по использованию DBI и обсуждается работа с данными в базе – извлечение данных, которые в ней уже хранятся, вставка новых данных, удаление и обновление существующих данных. Мы рассказываем о различных способах, которыми можно осуществлять эти операции, от простого этапа «заставить это работать» до более развитых и оптимизированных приемов работы с данными.

Глава 6 «Более сложные вопросы использования DBI».

Эта глава охватывает более сложные темы в сфере действия DBI, такие как тонкая настройка действия DBI в ваших программах, работа с типами данных LONG/LOB, метаданные команд и баз данных и, наконец, обработка транзакций.

Глава 7 «ODBC и DBI».

В этой главе обсуждается различие архитектур DBI и ODBC – другого переносимого API баз данных. И конечно, особое внимание уделено вопросу, почему программировать с помощью DBI проще.

Глава 8 «Командный процессор для DBI и прокси-доступ к базам данных».

В этой главе обсуждаются две темы, не имеющие непосредственного отношения к ядру DBI как таковому, но знание которых оказывается очень полезным. Сначала мы расскажем об оболочке DBI – инструменте командной строки, позволяющем подключаться к базам данных и делать произвольные запросы. Другая тема – архитектура доверительного доступа (proxy), с помощью которой DBI может, в частности, подключать сценарии на одной машине к базам данных на другой без необходимости установки программного обеспечения для сетевой работы с базами данных. Например, можно подключать сценарии, выполняющиеся на Unix-системе, к базе данных Microsoft Access, работающей на машине под Microsoft Windows.

Приложение A «Спецификация DBI».

Приложение содержит спецификацию DBI, поставляемую с `DBI.pm`.

Приложение B «Характеристики драйверов и баз данных».

В приложении содержатся полезные дополнительные данные обо всех часто используемых DBD и соответствующих им базах данных.

Приложение C «Хартия хранителей священных площадок ASLaN».

В этом приложении содержится Хартия сети хранителей древних святилищ (Ancient Sacred Landscape Network), имеющая целью сохранение древних священных мест, таких как мегалитические площадки, используемые для примеров в этой книге.

Дополнительные источники информации

Чтобы облегчить изучение некоторых тем книги, перечислим дополнительные источники, которые вам могут потребоваться до, во время или после чтения этой книги.

http://www.symbolstone.org/technology/perl/DBI

Домашняя страница DBI. Этот сервер содержит массу полезных сведений о DBI и о том, откуда получить различные модули. Здесь вы также найдете ссылки на активный список рассылки dbi-users и архивы.

http://www.perl.com/CPAN

Сервер объединяет Comprehensive Perl Archive Network – всестороннюю архивную сеть Perl, в которой можно найти массу полезных модулей, в том числе DBI.

C. J. Date «An Introduction to Database Systems»[1].

Стандартный учебник по базам данных, весьма рекомендуемый для чтения.

C. J. Date, Hugh Darwen «A Guide to the SQL Standard».

Отличная книга, подробная, но небольшая по объему и легко читаемая.

http://w3.one.net/~jhoffman/sqltut.htm

http://www.jcc.com/SQLPages/jccs_sql.htm

http://www.contrib.andrew.cmu.edu/~shadow/sql.html

На этих сайтах содержатся информация, спецификации и ссылки по языку запросов SQL, введение в который мы даем в главе 3. Дополнительную информацию можно получить, введя «SQL tutorial» или аналогичное выражение в строку поиска вашей любимой поисковой машины Интернета.

Randal Schwartz, Tom Christiansen «Learning Perl»[2].

Практическое руководство, цель которого помочь вам как можно быстрее начать писать сценарии Perl. Каждую главу сопровождают упражнения (с полными решениями). Большая новая глава знакомит вас с CGI-программированием, затрагивается также использование библиотечных модулей, ссылок и объектно-ориентированных конструкций в Perl.

Larry Wall, Tom Christiansen, Randal Schwartz «Programming Perl»[3].

Авторитетное руководство по Perl 5 – средству создания сценариев, ставшему предпочтительным средством программирования в World Wide Web, системном администрировании Unix и для большого числа других задач. Одним из авторов книги является Ларри Уолл (Larry Wall), создатель Perl.

Tom Christiansen, Nathan Torkington «The Perl Cookbook»[4].

Всеобъемлющее собрание задач, решений и практических примеров для всех, программирующих на Perl. Охватывает темы от вопросов, возникающих у новичков, до приемов, поучительных даже для опытных программистов на Perl. Нечто большее, чем простое собрание советов и ловких приемов, «The Perl Cookbook» является долгожданным дополнением к «Programming Perl», содержащим ранее не публиковавшиеся тайны Perl.

---

[1] К. Дейт «Введение в системы баз данных» 6 издание, изд-во «Диалектика».

[2] Р. Шварц, Т. Кристиансен «Изучаем Perl», изд-во «BHV-Киев».

[3] Л. Уолл, Т. Кристиансен, Р. Шварц «Программирование на Perl» 3 издание, изд-во «Символ-Плюс», февраль 2001 г.

[4] Т. Кристиансен, Н. Торкингтон «Perl: библиотека программиста», изд-во «Питер».

Lincoln Stein, Doug MacEachern «Writing Apache Modules with Perl and C».

В этой книге рассказывается, как расширить возможности вашего веб-сервера Apache, независимо от того, используете ли вы в качестве языка программирования Perl или C. В книге излагается архитектура Apache, mod_perl и Apache API. Что касается DBI, то в книге описывается модуль `Apache::DBI`, предоставляющий дополнительные развитые функции DBI для веб-сервисов, таких как создание пула постоянных соединений, оптимизированного для работы с базами данных через Сеть.

«Boutell FAQ» (*http://www.boutell.com/faq/*) и другие.

Эти ссылки имеют большую ценность для тех, кто собирается развертывать веб-сайты под управлением DBI. Обсуждаются общие вопросы, касающиеся того, как надо и как не надо реализовывать CGI-программы.

Randy Jay Yarger, George Reese, Tim King «MySQL & mSQL»[1].

Очень полезная книга для пользователей баз данных MySQL и mSQL. В ней рассказывается не только о самих этих базах данных, но и о драйверах DBI, а также других полезных темах, таких как CGI-программирование.

Типографские обозначения

В данной книге принято следующее использование шрифтов:

`Моноширинный`	Используется в названиях методов, в именах функций, переменных и атрибутов, а также в примерах исходных текстов программ.
Курсив	Используется в именах файлов и машин, в URL, а также для обозначения новых терминов.

Как с нами связаться

Мы опробовали и проверили весь материал книги, насколько было в наших силах, но вы можете столкнуться с тем, что какие-то функции изменились или в издание вкрались ошибки. Просим сообщить о замеченных вами ошибках и предложениях для будущих изданий по адресу:

O'Reilly & Associates

101 Morris Street

[1] Р. Яргер, Дж. Риз, Т. Кинг «MySQL и mSQL», изд-во «Символ-Плюс».

Sebastopol, CA 95472

800-998-9938 (из США или Канады)

707-829-0515 (международные или местные)

707-829-0104 (FAX)

Вы можете также послать нам сообщение электронной почтой. Чтобы быть включенным в наш лист почтовой рассылки или запросить каталог, посылайте письма по адресу:

info@oreilly.com

Вы можете задать технический вопрос или сообщить свои комментарии по поводу этой книги по адресу:

bookquestions@oreilly.com

У данной книги существует веб-сайт, где представлены примеры, ошибки и планы будущих изданий. Эту страницу можно найти на:

http://www.oreilly.com/catalog/perldbi

Дополнительные сведения об этой и других книгах можно найти на сервере O'Reilly:

http://www.oreilly.com

Примеры программ

Вы можете копировать тексты программ из книги и изменять их для собственных нужд. Однако вместо копирования вручную лучше загрузить код с *http://www.oreilly.com/catalog/perldbi*.

Благодарности

Аллигатор хочет поблагодарить свою жену Каролин за терпение к его авторской аффектации и метаниям во время написания книги. Имя Мартина Маккарти (Martin McCarthy) тоже должно быть упомянуто с благодарностью за многочисленные читки корректуры ранних набросков книги. Фил Кайзер (Phil Kizer) заслуживает благодарности за работу серверов, на которых сайт DBI располагался с 1995 до начала 1999 года. Карин и Джон Атвуд (Karin and John Attwood), Энди Бернхэм (Andy Burnham), Энди Норфолк (Andy Norfolk), Крис Твид (Chris Tweed) и многие другие члены списка рассылки stones заслуживают благодарности (и пива) за содействие по сохранению и представлению многих мегалитических площадок на территории Соединенного Королевства. Дополнительная благодарность всем, стоящим за ASLaN, за добровольное участие в трудной работе и хорошее ее исполнение.

Линда Мюи (Linda Mui) решительно заслуживает фирменного пакета O'Reilly и пары старых солнечных очков за изумительную работу по редактированию этой книги. И, наконец, с огромным чувством признательности я обращаюсь к Тиму, благодаря которому книга стала намного лучше, чем если бы я писал ее один.

Тим хотел бы поблагодарить свою жену Мари за то, что она его жена; Ларри Уолла за то, что он дал миру Perl; Теда Лемона (Ted Lemon) за идею, которая много лет спустя превратилась в DBI, и за долгие годы ведения списка рассылки. Спасибо также Тиму О'Рейлли (Tim O'Reilly) за то, что он извел меня требованиями написать книгу о DBI, и Аллигатору за то, что он фактически начал эту работу, а затем позволил мне присоединиться к ней (смирившись с моей склонностью к педантизму), и Линде Мюи за великолепное редактирование.

У DBI долгая история,[1] бесчисленное количество людей внесло свой вклад в его обсуждение и разработку за эти годы. Прежде всего, хотелось бы поблагодарить первопроходцев, в том числе Кевина Стока (Kevin Stock), Баццо Мочетти (Buzz Moschetti), Курта Андерсена (Kurt Andersen), Уильяма Хайлса (William Hails), Гарта Кеннеди (Garth Kennedy), Майкла Пепплера (Michael Peppler), Нейла Бриско (Neil Briscoe), Дэвида Хьюза (David Hughes), Джеффа Стэндера (Jeff Stander) и Форреста Д. Уитчера (Forrest D. Whitcher).

И, конечно, нужно поблагодарить тех несчастных, которые прорывались сквозь бессчетные недокументированные препятствия, чтобы реализовать на практике драйверы DBI. В их рядах находятся Йохан Видман (Jochen Wiedmann), Аллигатор Декарт (Alligator Descartes), Джонатан Леффлер (Jonathan Leffler), Джефф Арлвин (Jeff Urlwin), Майкл Пепплер (Michael Peppler), Хенрик Тугар (Henrik Tougaard), Эдвин Пратомо (Edwin Pratomo), Дэвид Майглваккей (Davide Migliavacca), Ян Паздзиора (Jan Pazdziora), Питер Хэворс (Peter Haworth), Эдмунд Мергл (Edmund Mergl), Стив Уильямс (Steve Williams), Томас Лауэри (Thomas Lowery) и Филип Пламли (Phlip Plumlee). Без них DBI не смог бы стать сегодня реальностью.

Оба мы хотим поблагодарить многочисленных рецензентов, сделавших ценные замечания. Особая благодарность Мэтью Персико (Matthew Persico), Натану Торкингтону (Nathan Torkington), Джеффу Роуи (Jeff Rowe), Деннису Годдарду (Denis Goddard), Хонцу Паздзиора (Honza Pazdziora), Ричу Миллеру (Rich Miller), Ниаму Кеннеди (Niamh Kennedy), Рэнделу Шварцу (Randal Schwartz) и Джеффри Бэйкеру (Jeffrey Baker).

[1] Все это началось 29 сентября 1992 года.

1

Введение

«Базы данных» – большая и сложная тема, охватывающая множество различных понятий структуры, вида и предполагаемого использования. Существует также множество разнообразных способов доступа к данным, хранимым в этих базах, и воздействия на них.

В этой книге описывается интерфейс баз данных для Perl, называемый DBI и предоставляющий унифицированный интерфейс для доступа к данным, хранящимся во многих различных базах. DBI позволяет писать на Perl программы, осуществляющие доступ к данным, без необходимости учитывать специфические для базы данных или платформы вопросы или фирменные интерфейсы.

Мы также рассмотрим способы хранения, извлечения и обработки данных в Perl без применения DBI, поскольку нередки ситуации, когда использование базы данных не является насущной необходимостью, но, тем не менее, требуется некая форма структурированного хранения данных.

Для начала мы обсудим некоторые наиболее частые применения систем баз данных в деловой жизни и место, которое занимают Perl и DBI в этих рамках.

От мэйнфреймов к рабочим станциям

В настоящее время базы данных повсеместно используются в вычислительной практике. В прежние годы они почти исключительно использовались в вычислительных средах мэйнфреймов. Сейчас, когда компьютеры размером с коробку для обуви обладают большей вычислительной мощностью, чем прежние машины, занимавшие целую

комнату, высокопроизводительная обработка баз данных доступна каждому.

Помимо более дешевых и более мощных аппаратных устройств стали доступны меньшие пакеты баз данных, такие как Microsoft Access и mSQL. Эти пакеты позволяют любому пользователю компьютера применять мощную технологию баз данных в повседневной жизни.

В корпоративной рабочей среде также наблюдается резкая децентрализация ресурсов баз данных с коренным уменьшением масштабов операций в некоторых фирмах, в ходе которой производится замена баз данных на мэйнфреймах набором небольших баз данных, распределенных по персональным компьютерам и рабочим станциям. В результате разработчики и пользователи часто становятся ответственными за администрирование и поддержку собственных баз и наборов данных.

Тенденция к смешению и согласованию технологий баз данных имеет целый ряд существенных недостатков. Так, заменив централизованную базу данных кластером рабочих станций и несколькими типами баз данных, компании столкнулись с задачей найма новых квалифицированных администраторов или переобучения уже имеющихся. Кроме того, администраторам приходится теперь изучать, как объединять вместе различные типы баз данных.

В этой обстановке возник новый тип техники программного обеспечения, а именно – *независимые от баз данных* программные интерфейсы. Если уменьшение масштабов создало проблемы для административного персонала, то еще большее влияние оно, возможно, оказало на разработчиков.

В централизованной среде мэйнфрейма предполагается, что программное обеспечение баз данных пишется на стандартном языке, возможно, COBOL или C, и выполняется только на одной машине. В распределенной же среде могут поддерживаться несколько баз данных с различными операционными системами и процессорами, причем каждая команда разработчиков выбирает собственную предпочтительную среду разработки (такую как Visual Basic, PowerBuilder, Oracle Pro*C, Informix E/SQL, C++ с ODBC – список почти бесконечен). Поэтому задача координации и переноса программного обеспечения быстро превратилась из достаточно простой в крайне сложную.

Независимые от баз данных программные интерфейсы пришли на помощь несчастным осажденным разработчикам, снабдив их единым унифицированным интерфейсом, с помощью которого они могут программировать. Они ограждают разработчиков от необходимости изучения каждого типа баз данных, с которым они работают, и позволяют с большей легкостью переносить программное обеспечение с одного типа баз данных на другой. Например, программное обеспечение, написанное первоначально для базы данных на мэйнфрейме, с небольшими модификациями сможет работать с базой данных Oracle. Программное

обеспечение, написанное для Informix, как правило, с небольшими изменениями работает под Oracle. А программное обеспечение, написанное для Microsoft Access, с помощью небольших модификаций будет работать на Sybase.

Если соединить этот независимый от базы данных программный интерфейс с таким языком программирования, как Perl, нейтральным в отношении операционной системы, то появляется перспектива снова получить единую базу программного кода. Как в старые добрые времена, но с большой разницей: теперь вы полностью владеете мощью распределенных баз данных.

Независимые от баз данных программные интерфейсы помогают не только разработчикам. Администраторы могут использовать их для быстрого написания переносимых средств контроля и администрирования баз данных, увеличивая как свою производительность, так и производительность систем и баз данных, за мониторинг которых они отвечают. Этот процесс приводит к лучшей настройке систем и большей их доступности, предоставляя администраторам возможность активной поддержки обслуживаемых систем.

Другая сторона современного стиля ведения корпоративных баз данных касается идеи *хранилищ данных* (*data warehousing*), что означает создание больших хранилищ архивных данных, в которых можно осуществлять просмотр или поиск информации отдельно от активных баз данных. Мощные языки высокого уровня с независимыми от баз данных программными интерфейсами, такие как Perl, приобретают большое значение в создании и сопровождении хранилищ данных. Это связано не только с их способностью без проблем перемещать данные из одной базы в другую, но и с возможностью эффективно просматривать, упорядочивать, преобразовывать и обрабатывать данные.

В целом, базы данных приобретают все большее значение в корпоративной среде, и для того чтобы не допустить раскола этих данных на отдельные местные фрагменты, необходимы мощные интерфейсы. Этот процесс склеивания отдельных кусков может быть облегчен в результате использования программных интерфейсов, независимых от баз данных, таких как DBI, особенно в сочетании с эффективными языками высокого уровня, подобными Perl.

Perl

Perl является языком очень высокого уровня, первоначально разработанным в 1980-х годах Ларри Уоллом (Larry Wall). Сейчас Perl разрабатывается под бдительным оком Ларри группой людей, известных как Perl5-Porters. Одной из многих мощных возможностей Perl является способность обрабатывать произвольные фрагменты текстовых данных, известных как *strings* (строки), разнообразными способами, включая обработку регулярных выражений. Эта возможность де-

лает Perl отличным выбором для программирования баз данных, поскольку большая часть хранящейся в базах данных информации имеет текстовую природу. В отличие от C, не очень хорошо приспособленного для обработки строк, Perl устраняет из процесса программирования эти трудности. Сценарии на Perl оказываются значительно меньше, чем эквивалентные программы на C, и обычно переносимы на другие операционные системы, где программы Perl выполняются после небольших модификаций или вообще без них.

В настоящее время Perl получил возможность динамической загрузки внешних *модулей*, представляющих собой части программного обеспечения, которые могут быть вставлены в Perl для расширения его функциональности. Сейчас таких модулей буквально сотни, от математических до модулей передачи трехмерной графики и модулей, позволяющих взаимодействовать с сетями и сетевым программным обеспечением. DBI является набором модулей для Perl, позволяющих взаимодействовать с базами данных.

В последние годы использование Perl превратилось в стандарт для многих фирм благодаря исключительной полезности его во многих различных приложениях, как своего рода «швейцарского солдатского ножа» языков программирования. Он активно используется системными администраторами, которым нравится его гибкость и пригодность для почти любых мыслимых задач. В сочетании с DBI Perl существенно упрощает загрузку и выгрузку баз данных, а его отличные способности обработки данных позволяют разработчикам с легкостью создавать и обрабатывать данные.

Более того, Perl молчаливо принят de facto как язык для создания CGI-приложений в Сети. Какое отношение это имеет к DBI? С помощью Perl и DBI можно быстро создавать мощные CGI-сценарии, генерирующие динамические веб-страницы из данных, содержащихся в базах данных. Например, каталоги электронной торговли могут храниться в базе данных и представляться покупателям как ряд динамически создаваемых веб-страниц. Примеры программ в этой книге используют базу данных археологических площадок, которую можно разместить в Интернете.

Исходя из такого подтверждения концепции, а также появления новых мощных модулей, таких как DBI и средства быстрой разработки графических интерфейсов пользователя Tk, крупные компании сейчас рассчитывают на то, что Perl обеспечит возможность быстрой разработки эффективных, надежных и переносимых приложений, которые могут быть развернуты в корпоративных сетях интранет и в Интернете.

DBI в реальном мире

Сегодня DBI используется во всем мире многими компаниями, такими как NASA и Motorola, в том числе в крупномасштабных критически важных средах. Приведем примеры свидетельств активных пользователей DBI со всего света:

> Мы разработали для одного своего крупного клиента крупномасштабную систему регистрации и анализа телефонных звонков и поддерживаем ее. Система ежедневно собирает около 1 Гбайт данных по звонкам примерно с 1 200 000 телефонных номеров. К данному времени обработано около 424 Гбайт данных (свыше 6 200 000 000 звонков). Данные обрабатываются и загружаются в Oracle с помощью DBI и DBD::Oracle. В базе данных содержатся сведения примерно о 20 миллионах звонков, которые можно просмотреть. Система создает ежемесячно около 44 000 высококачественных постскриптовских отчетов (около пяти страниц с одиннадцатью цветными графиками и пятью таблицами), получаемых заполнением шаблонов FrameMaker с помощью Perl. (Цифры верны на июль 1999 года и неуклонно растут.)
>
> Система развернута на трех двухпроцессорных машинах Sun SPARC Ultra 2. Одна используется для получения и обработки данных, на второй установлена Oracle, а третья осуществляет главную работу по генерации отчетов, распределенную также и по двум другим машинам. Система почти полностью реализована на Perl.
>
> Имеется лишь одна программа, написанная не на Perl, и то только потому, что она была написана раньше, и не является специфичной для системы. Прочий код, написанный не на Perl, состоит из нескольких небольших библиотек, подключенных к Perl через интерфейс XS.
>
> Цитата из подведения итогов проекта старшим управляющим: «Менее чем через год служба заработала. Это событие было отмечено как один из наиболее быстро прошедших от замысла до запуска проектов такого размера и сложности.»
>
> Система спроектирована, разработана, реализована, инсталлирована и поддерживается Paul Ingram Group, получившей за участие в проекте награду «Rising to the Challenge» («Принявший вызов»). Без Perl система не смогла бы быть разработана в течение требуемых очень сжатых сроков. Опять же, без Perl систему нельзя было бы так легко сопровождать или быстро расширять для удовлетворения изменяющихся требований.
>
> *Тим Банс (Tim Bunce), Paul Ingram Group*

> В 1997 году я разработал для Исследовательского центра NASA в Лэнгли, штат Виргиния, систему, предоставляющую интерфейс для поиска из Сети данных в базе, содержащей сведения о примерно 100 000 элементов оборудования, принадлежащих NASA. Я использовал Apache, DBI, Informix, WDB и mod_perl на Sparc 20. Работала, как в сказке. Она так понравилась в NASA, что ее использовали для демонстраций на совещаниях по реорганизации аэродинамических труб. Дело обстояло таким образом, что каждый раз после показа системы другим людям мне приходилось расширять ее, добавляя вещи, подобные отслежива-

нию ремонтируемого оборудования или выводу графических изображений оборудования, так что когда они теряли листы спецификации, то могли посмотреть их электронный вариант. Когда все работает, успех способствует успеху.

Джефф Роуи (Jeff Rowe)

Я работаю над системой, реализованной с помощью Perl, DBI, Apache (mod_perl) на машине с Red Hat Linux 5.1, с использованием MySQL – облегченной SQL РСУБД.[1] Система предназначена для работы в крупной международной холдинговой компании, владеющей примерно 50 другими компаниями. В компании занято около 30 000 служащих по всему миру, которым нужна надежная система для доступа к веб-ресурсам. Этот первый цикл интранет рассчитан на обработку до сорока запросов веб-объектов в секунду (примерно 200 одновременно работающих пользователей) и работает на однопроцессорном Intel Pentium Pro с 512 Мбайт памяти. Мы ведем разработку на Perl, всюду используя объектно-ориентированную технологию. За последнюю пару лет мы разработали большую библиотеку повторно используемого кода на Perl. Один из наших наиболее полезных модулей создает объектно-реляционную оболочку вокруг DBI, что позволяет нашим разработчикам приложений обращаться к базе данных с использованием объектно-ориентированных методов для доступа к записям или изменения их свойств. Мы сэкономили несчетное количество времени и денег, выбрав Perl вместо более частной системы.

Джесси Эрлбаум (Jesse Erlbaum)

Motorola Commercial Government and Industrial Systems использует Perl вместе с DBI и DBD-Oracle как часть ориентированной на Интернет системы отчетности для большого числа производящих и распределяющих организаций. Использование DBI/DBD-Oracle является составной частью отхода от систем отчетности, базирующихся на Oracle Forms, к платформе, базирующейся чисто на веб. В эксплуатации находится несколько приложений среднего размера, основанных на DBI, включая простые приложения распространения извещений, динамическую маршрутизацию подтверждений и важные бизнес-приложения. Хотя требуется несколько большее «терпение» для разработки веб-приложений и «хорошо выглядящих» пользовательских интерфейсов, мой опыт показывает, что для реализации приложений с использованием DBI требуется несколько меньшее время, чем при альтернативных вариантах. Сроки «исправления» использующих DBI/DBD программ также оказываются короче. Качество программ при использовании подхода DBI/DBD оказывается выше, но это может объясняться различиями в методологии разработки программного обеспечения.

Гарт Кеннеди (Garth Kennedy), Motorola

[1] РСУБД – система управления реляционными базами данных.

Историческое отступление и стоящие камни

По всей книге разбросаны примеры, связанные с обсуждаемыми темами. Для того чтобы не дать примерам сбить вас с толку еще больше, чем это сделает текст, обсудим заранее данные, которыми мы будем манипулировать в примерах.

Преимущественно в Великобритании, но также и в других странах по всему миру есть сооружения из каменных глыб, или *мегалиты*.[1] Камни располагаются кольцами, рядами, одиночно или попарно. Никто не может с полной уверенностью сказать, каково назначение этих сооружений, но существует масса теорий от уклончивого «ритуального» использования до более решительной теории посадочных площадок инопланетян. Наиболее известное и часто посещаемое из этих сооружений – Стоунхендж, расположенный на юге Англии. Однако Стоунхендж является уникальным и нетипичным мегалитическим монументом.

Недостаточное понимание происхождения мегалитов отчасти объясняется тем, что возраст их насчитывает до 5 тысяч лет. Просто не существует письменных источников, описывающих назначение, ритуалы или разумные основания возведения этих сооружений. Однако существует много сайтов, излагающих различные теории их происхождения.

Примеры программ, приведенные в данной книге, и образцы веб-приложений, которые мы также предоставляем, используют базу данных, содержащую сведения об этих сооружениях.

[1] От греческого «большой камень». Для многих сооружений это может быть неудачным названием, поскольку образующие круг камни оказываются не более полуметра в высоту. Однако в некоторых экстремальных случаях, таких как Стоунхендж и Эйвбери, приставка «мега» более чем оправдана.

2

Основные базы данных, не использующие DBI

Существует несколько способов организации данных, содержащихся в базах данных. Наиболее распространенным из них является технология *реляционных баз данных*. Базы данных, использующие реляционную модель, называются *системами управления реляционными базами данных,* или РСУБД. Наиболее популярные в настоящее время системы баз данных (такие как Oracle, Informix и Sybase) являются реляционными по своей архитектуре.

Но что в действительности означает «реляционная»? Реляционной базой данных называется такая база данных, которая воспринимается пользователем как набор таблиц, при этом таблица является неупорядоченным набором строк. (Допуская вольность речи, отношение (relation) является просто математическим термином для такой таблицы.) В каждой строке таблицы содержится фиксированное число полей, и в каждом поле может храниться предопределенный тип данных, например целое, дата или строка.

Другим видом методологии, приобретающим все большее распространение, является объектно-ориентированная методология, или ООСУБД. В объектно-ориентированной модели все, что входит в базу данных, рассматривается как объекты, относящиеся к некоторым классам, внутри которых определены правила обработки заключенных в них данных. Эта методология близко смыкается с объектно-ориентированными языками, такими как Smalltalk, C++ и Java. Однако DBI не поддерживает никакой реально существующей ООСУБД, поэтому в данное время мы не станем углубляться в эту методологию.

Наконец, существует несколько пакетов упрощенных баз данных, поддерживаемых в различных операционных системах. Эти пакеты

простых баз данных, как правило, не обладают теми изощренными функциями, которые предоставляют ядра «настоящих» баз данных. Фактически они являются лишь несколько усложненными программами обработки файлов, а не настоящими пакетами для работы с базами данных. Однако в их пользу говорит то, что они могут работать чрезвычайно быстро, а в некоторых ситуациях усложненные функции, предоставляемые «настоящими» базами данных, являются просто лишней нагрузкой.[1]

В этой главе мы расскажем о некоторых базах данных, с которыми DBI не используется, от простейших файлов данных ASCII до дисковых хэш-файлов с поддержкой дубликатов ключей. Попутно мы обсудим проблемы одновременного доступа и блокировки и некоторые применения довольно полезных модулей `Storable` и `Data::Dumper`. (Хотя эти темы не связаны с DBI непосредственно, мы полагаем, что их освещение может для многих оказаться полезным, и даже ветераны DBI смогут найти несколько удобных приемов.)

Все эти технологии баз данных, от самых сложных до простейших, имеют два основных свойства. Первое состоит в самом определении термина: база данных – это набор хранимых в компьютере данных, поверх которого находятся различные уровни абстракции. Каждый уровень абстракции обычно упрощает организацию хранимых данных и доступ к ним благодаря тому, что разделяются запрос для получения конкретных данных и механизм их получения.

Вторым основным свойством всех систем баз данных является то, что во всех них для получения доступа к хранимым данным используются интерфейсы прикладного программирования (API). В простейших базах данных API состоит просто из вызова функций операционной системы для чтения/записи файлов посредством вашего любимого языка программирования.

API позволяет программистам взаимодействовать со сложными программными модулями по правилам доступа, определенным создателями исходного программного обеспечения. Хорошим примером этого служит Berkeley Database Manager API. Помимо простого доступа к данным этот API позволяет изменять структуру базы данных и сами хранимые в ней данные. Такой более высокий уровень доступа к базе данных дает то преимущество, что не нужно беспокоиться о том, *как* Berkeley Database Manager управляет данными. Вы работаете с абстрагированным представлением через API.

На более высоких уровнях доступа, таких как реализуемые в РСУБД, API доступа и обработки данных полностью отделен от структуры базы данных. Это разделение логической модели и физического пред-

[1] Полезный список свободно распространяемых баз данных имеется на *ftp://ftp.idiom.com/pub/free-databases*.

ставления позволяет писать стандартный код для работы с базой данных, например на SQL, который независим от используемого ядра базы данных.

Администраторы хранения данных и слои программного обеспечения

Современные базы данных, независимо от реализованной в них методологии, обычно состоят из нескольких слоев программного обеспечения. Каждый слой реализует более высокий уровень функциональности, используя интерфейсы и сервисы, имеющиеся в слоях более низкого уровня.

Например, одноуровневые базы данных состоят из пулов данных и очень немногочисленных слоев абстрагирования. Базы данных такого типа позволяют обрабатывать хранимые в них данные посредством прямого изменения того, как эти данные хранятся в самих файлах. Тем самым достигается большая мощь и гибкость ценой трудности использования, ограниченных функциональных возможностей и нервного напряжения, вызываемого отсутствием средств страховки. Вся работа с файлами данных происходит путем использования стандартных файловых операций Perl, в свою очередь, основанных на API операционной системы.

Файловые библиотеки DBM, такие как Berkeley DB, служат примером слоя администратора хранения данных (storage manager), лежащего поверх файлов чистых данных и позволяющего обрабатывать хранимые в базе данные через четко определенный API. Этот администратор хранения данных транслирует вызовы API в операции обработки данных в интересах пользователя, препятствуя непосредственному изменению пользователем структуры данных, в результате которого данные могли бы оказаться поврежденными и нечитаемыми. Управление базой данных через такого администратора хранения данных значительно проще и безопаснее, чем непосредственное ее изменение.

Можно реализовать поверх файлов DBM более мощную систему базы данных. Этот новый слой использовал бы DBM API для реализации более мощных функций и создал бы еще один уровень абстрагирования между пользователем и реальными физическими файлами, содержащими данные.

Использование администраторов более высокого уровня предоставляет многие преимущества. Уровни абстрагирования между программой пользователя и базой данных позволяют производителям баз данных прозрачным образом улучшать оптимизацию, изменять структуру файлов базы данных или осуществлять перенос ядра базы данных на другие платформы, при котором не возникает необходимости в изменении прикладных программ.

Языки запросов и функции данных

Операции работы с базой данных можно разделить на те, в которых производится воздействие на саму базу данных (т. е. логическую и физическую организацию составляющих ее файлов), и те, в которых производится воздействие на сами данные, хранимые в этих файлах. Первая группа в целом специфична для конкретной базы данных и может быть реализована многими способами, а вторая группа обычно выполняется с использованием *языка запросов*.[1]

Все языки запросов от нижнего уровня использования функций Perl для обработки строк и чисел до языка запросов высокого уровня, такого как SQL, реализуют четыре главные операции, с помощью которых обрабатываются данные:

Выборка

Чаще всего работа с базой данных состоит в извлечении хранящихся в ней данных. Эта операция известна как *выборка* (*fetching*). Она возвращает необходимые данные в виде, понятном для базового языка API, используемого для создания запросов к базе данных. Например, при необходимости использовать Perl для запроса данных из базы Oracle эти данные запрашиваются с использованием языка запросов SQL, а возвращаемые данные должны иметь вид строк и чисел Perl. Эта операция известна также как *выбор* (*selecting*), от ключевого слова SQL `SELECT`, используемого при выборке данных из базы.

Сохранение

Для того чтобы данные могли быть выбраны, сначала должно быть произведено их *сохранение* (*storing*). Слой администратора хранения данных переводит величины, представленные на языке программирования, в такие, которые понятны базе данных. После этого администраторы хранения данных записывают эти величины в файлы данных. Эта операция известна также как *вставка* (*inserting*) данных.

Обновление

Если данные записаны в базу, это не значит, что они остаются неизменными. При необходимости их можно изменить. Например, в базе данных, хранящей данные о товарах, со временем могут изменяться сведения о ценах для каждого продукта. Операция изменения значения данных, уже существующих в базе, называется *обновлением* (*updating*). Важно отметить, что при этой операции к

[1] Мы употребляем термин «язык запросов» в очень широком смысле. Мы охватываем им все – от вербальных командных языков типа SQL до жестко закодированной логики в таких языках программирования, как Perl.

базе данных не добавляются новые элементы и не удаляются старые, но изменяются уже существующие.[1]

Удаление

Последней базовой операцией, которую обычно требуется производить с данными, является *удаление* из базы старых или избыточных данных. При этой операции данные полностью удаляются из базы с помощью администратора хранения, вырезающего данные из файлов. После удаления из базы данные нельзя восстановить или заменить, кроме как заново вставив в базу.[2]

Эти операции часто обозначают акронимом C.R.U.D. (Create, Read, Update, Delete). В данной книге эти темы обсуждаются в несколько ином порядке, в основном потому, что, на наш взгляд, большинство читателей, по крайней мере вначале, будет заниматься извлечением данных из существующих баз, а не созданием новых баз для хранения данных.

Стоящие камни и учебная база данных

Наши небольшие учебные базы данных, встречающиеся в этой главе, содержат данные о мегалитах в Великобритании. В последующих главах используется более сложная версия этой базы данных.

В число основных данных о мегалитах[3], которые мы хотим хранить, входят название, местонахождение в стране, уникальный картографический код, тип мегалита (например, круг камней или стоящий камень) и описание того, как выглядит площадка.

Например, мы хотим сохранить в базе данных следующие сведения о Стоунхендже:

Name:

Стоунхендж.

Location:

Уилтшир, Англия.

Map Reference:

SU 123 400.

---

[1] На логическом уровне это так, но на физическом обновление может осуществляться путем удалений и вставок.

[2] Если только вы не используете *транзакции*. Подробнее о них говорится в главе 6.

[3] Хранение чего-либо *на* мегалите является прямым нарушением принципов, изложенных в приложении C. Это замечание приводится на тот случай, если вы пропустили знакомство с мегалитами, представленное нами в главе 1.

Type:

Каменный круг и хендж.

Description:

Самый знаменитый в мире мегалит, образованный земляным валом, или *хенджем*, и несколькими концентрическими кольцами массивных стоящих камней – *дольменами*.

Из этой учебной базы мы можем извлекать различные сведения, например, потребовать данные о всех мегалитах в Уилтшире или о всех стоящих камнях в Оркни и т. д.

Теперь обсудим простейший вид базы данных, который можно использовать, – *одноуровневую базу данных* или *базу данных в плоском файле*.

Одноуровневые базы данных

Простейшим типом базы данных, которую мы можем создать для работы, является старая проверенная *одноуровневая база данных* (*flat-file database*). Эта база данных является, по существу, файлом или группой файлов, содержащих данные в известном стандартном формате, которые программа просматривает в поисках требуемых данных. Модифицирование данных обычно производится путем обновления находящейся в памяти копии данных, которые содержатся в одном или нескольких файлах, с последующей обратной записью на диск всего набора данных. Одноуровневые базы данных обычно являются текстовыми ASCII-файлами, каждая строка которых содержит одну запись данных. Признак конца строки служит разделителем записей.

В этом разделе мы рассмотрим два основных типа баз данных в плоских файлах: файлы, в которых поля отделяются одно от другого символом-разделителем, и файлы, в которых каждое поле имеет фиксированную длину. Мы обсудим преимущества и недостатки каждого типа файла данных и представим примеры программ для работы с ними.

По-видимому, чаще всего одноуровневые базы данных используют формат файла *с разделителем*, в котором все поля отделяются одно от другого символом-разделителем. И, вероятно, наиболее распространенным из этих форматов является файл с *разделителем-запятой* (*comma-separated values* – CSV), в котором поля отделяются одно от другого запятыми. Этот формат воспринимают многие распространенные программы, такие как Microsoft Access и электронные таблицы. Благодаря этому данный формат является отличным переносимым форматом базового уровня, который удобно использовать при совместном доступе к данным из различных приложений.[1]

Другими часто используемыми символами-разделителями являются двоеточие (:), символ табуляции и символ вертикальной черты (|).

Примером файла с разделителями является файл */etc/passwd* операционной системы Unix, в котором разделителем служит двоеточие. На рис. 2.1 показана одна запись из файла */etc/passwd*.

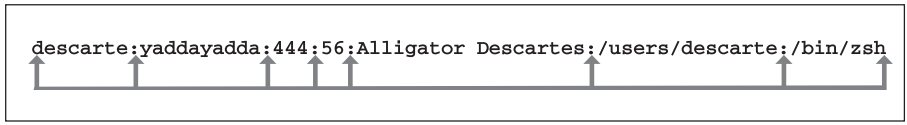

***Рис. 2.1.** Формат записи файла /etc/passwd*

Запросы данных

Поскольку файлы с разделителем относятся к очень низкому уровню администратора хранения данных, все действия с данными должны производиться при помощи функций операционной системы и очень низкоуровневой логики запросов, такой как базовые операции сравнения строк. В следующей программе продемонстрированы открытие файла с разделителем-двоеточием, содержащего данные о мегалитах, поиск нужной площадки и возврат найденных данных:

```
#!/usr/bin/perl -w
#
# ch02/scanmegadata/scanmegadata: Просматривает заданный файл с данными
#                                 по мегалитам в поисках нужной площадки.
#                                Использует двоеточие для разделения полей.
#

### Проверить, задал ли пользователь следующие аргументы:
###     1) Имя файла, содержащего данные
###     2) Название площадки, которую нужно найти
die "Использование: scanmegadata <файл данных> <название площадки>\n"
    unless @ARGV == 2;

my $megalithFile = $ARGV[0];
my $siteName     = $ARGV[1];

### Открытие файла данных для чтения или аварийный выход
open MEGADATA, "<$megalithFile"
    or die "Невозможно открыть $megalithFile: $!\n";

### Объявить переменные полей в строке
my ( $name, $location, $mapref, $type, $description );

### Объявить флаг 'запись найдена'
```

[1] Еще более замечательно, что есть драйвер DBI под названием DBD::CSV, позволяющий писать SQL-команды для обработки данных в плоских файлах формата CSV.

```
my $found;

### Просмотреть все записи в поиске нужной площадки
while ( <MEGADATA> ) {

    ### Удалить символ новой строки, являющийся разделителем записей
    chop;

    ### Разбить запись на отдельные поля
    ( $name, $location, $mapref, $type, $description ) =
         split( /:/, $_ );

    ### Сравнить название площадки с содержащимся в записи
    if ( $name eq $siteName ) {
         $found = $.;  # $. содержит номер текущей строки файла
         last;
    }
}

### Если найдена нужная площадка, вывести данные о ней
if ( $found ) {
     print "Найдена площадка: $name в строке $found\n\n";
     print "Данные по площадке $name ( $type )\n";
     print "===============",
         ( "=" x ( length($name) + length($type) + 5 ) ), "\n";
     print "Местонахождение:      $location\n";
     print "Код карты: $mapref\n";
     print "Описание:   $description\n";
}

### Закрыть файл с данными по мегалитам
close MEGADATA;

exit;
```

Например, если запустить эту программу с файлом, содержащим запись следующего вида:[1]

```
Стоунхендж:Уилтшир:SU 123 400:Каменный круг и хендж:Самый известный
каменный круг
```

и искать термин `Стоунхендж`**, то будут выведены следующие данные:**

```
Найдена площадка: Стоунхендж в строке 1

Данные по площадке Стоунхендж (Каменный круг и хендж)
==================================================
Местонахождение: Уилтшир
```

[1] **В этом примере, как и в нескольких последующих, одна строка разбита на две только для того, чтобы уместиться на странице книги.**

```
Код карты:      SU 123 400
Описание:       Самый известный каменный круг
```

указывающие на то, что наш поиск площадки методом грубой силы работает. Пример программы показывает, что мы использовали собственные функции Perl файлового ввода/вывода и собственные функции Perl обработки строк для расщепления данных с разделителем и проверки их на соответствие критерию поиска.

Недостаток использования файлов с разделителем заключается в том, что если символ-разделитель содержится в самих данных, необходимо проявлять осторожность, чтобы не разбить запись в неверном месте. Использование функции Perl `split()` с простым регулярным выражением в том виде, как это сделано в примере, не принимает этого во внимание и может привести к неверным результатам. Например, следующая запись приведет к тому, что при использовании `split()` мы получим не то, что требуется:

```
Стоунхендж:Уилтшир:SU 123 400:Каменный круг и хендж:Стоунхендж: Самый
известный каменный круг
```

Простейшим способом наскоро исправить положение является замена символа-разделителя, содержащегося в данных, каким-либо другим символом, относительно которого вы уверены, что в данных он встретиться не может. При этом нужно не забыть сделать обратное преобразование при выборке данных.

Другим распространенным способом хранения данных в плоских файлах является использование *записей фиксированной длины*. Это подразумевает помещение каждого элемента данных в поле точно определенного размера в файле данных. При таком формате не требуется использование символа-разделителя между полями. Не требуется также и разделять записи между собой, но мы продолжим использовать в наших примерах символ конца строки для разграничения записей, поскольку в Perl очень легко обрабатывать записи построчно.

Использование полей фиксированной длины напоминает организацию данных в более мощных системах баз данных, таких как РСУБД. Благодаря предварительному выделению пространства для данных администратор хранения данных может делать предположения о размещении данных на диске и производить соответствующую оптимизацию. Для наших мегалитических данных можно установить такие размеры полей:[1]

[1] Тот факт, что каждый размер является степенью двух, имеет лишь то значение, что указывает на почтенный возраст авторов, которые помнят те времена, когда использование степени двух в некоторых случаях оказывалось полезным. Как правило, это больше не является существенным.

```
Поле            Размер в байтах
-----           ---------------
Name            64
Location        64
Map Reference   16
Type            32
Description     256
```

Для хранения данных в таком формате администратор хранения должен использовать несколько иную логику, хотя стандартные функции файлового ввода/вывода Perl по-прежнему можно применять. Для проверки наличия в записи нужных данных необходимо реализовать иной способ извлечения полей. Для файлов с полями фиксированной длины отлично подходит функция Perl `unpack()`. В следующем фрагменте программного кода показано, как функция `unpack()` заменяет использовавшуюся выше `split()`:

```
### Разбить запись на отдельные поля
### с учетом перечисленных выше размеров данных
( $name, $location, $mapref, $type, $description ) =
    unpack( "A64 A64 A16 A32 A256", $_ );
```

Поля фиксированной длины имеют постоянный размер, но помещаемые в них данные могут иметь размер, не совпадающий с размером поля. В этом случае свободное пространство заполняется символом, который обычно в данных не встречается или который можно проигнорировать. Обычно это символ пробела (ASCII 32) или `nul` (ASCII 0).

В нашем примере считается, что данные упакованы с дополнением пробелами, поэтому мы удаляем концевые пробелы из поля, чтобы они не мешали поиску. Это можно сделать, используя в `unpack()` формат с `A` в верхнем регистре.

Если приходится выбирать между полями с символом-разделителем и полями с фиксированной длиной, нужно учитывать следующее:

Основные ограничения

Главное ограничение при использовании символа-разделителя заключается в необходимости добавления специальной обработки символов конца записи и границы полей в данных, чтобы избежать их попадания в значение поля.

Главным ограничением полей фиксированной длины является фиксированная длина. Нужно проверять длину записываемых значений, чтобы она не оказалась слишком велика (или разрешить молчаливое усечение этих значений). Если нужно увеличить размер поля, приходится создавать специальную утилиту для переписывания файла в новом формате, а также просматривать и исправлять все сценарии, непосредственно работающие с файлом.

Занимаемое пространство

Файл с использованием разделителей часто занимает меньше места, чем файл с записями фиксированной длины, хранящий те же данные, причем иногда *значительно меньше*. Все зависит от размера пустых или частично заполненных полей и их количества. Например, некоторые значения полей, такие как URL в Интернете, могут быть очень длинными, но обычно они очень коротки. При хранении их в длинных полях фиксированного размера напрасно расходуется большое количество дискового пространства.

Хотя файлы с разделителями часто занимают меньше места, они все же непроизводительно тратят пространство на все символы-разделители. Если нужно хранить большое число очень маленьких полей, то предпочтение может быть отдано записям фиксированной длины.

Быстродействие

В наши дни вычислительные мощности растут быстрее, чем скорости обмена данными с жесткими дисками. Иными словами, иногда выгоднее использовать более эффективные методы хранения данных, даже если это приводит к большим затратам процессорного времени.

Обычно для последовательного доступа лучше использовать файлы с разделителями, чем файлы с записями фиксированной длины, поскольку уменьшение размера более чем компенсирует увеличение расхода процессорного времени, вызванное необходимостью обработки замененных в тексте символов-разделителей.

Однако у файлов с записями фиксированной длины есть свой козырь: прямой доступ. При желании выбрать из файла с разделителями запись номер 42927 вам *необходимо* читать весь файл и вести подсчет записей, пока вы не доберетесь до нужной. При использовании файла с записями фиксированной длины можно просто умножить 42927 на суммарную длину записи и перейти на нее за один шаг с помощью функции `seek()`.

Более того, если запись найдена, ее можно обновить прямо на месте, записав в нее новые данные. Поскольку новая запись имеет тот же размер, что и прежняя, нет опасности повредить следующую запись.

Вставка данных

Вставлять данные в одноуровневую базу очень просто, и обычно для этого требуется только присоединить новые данные к концу файла. Например, вставка новой записи о мегалите в файл с разделителем-двоеточием может быть осуществлена так:

```
#!/usr/bin/perl -w
#
# ch02/insertmegadata/insertmegadata: Вставляет новую запись в файл
#                                     с данными о мегалитах, используя
#                                     двоеточие в качестве разделителя
#

### Проверить, задал ли пользователь аргументы
###     1) Имя файла, содержащего данные
###     2) Название площадки, данные о которой нужно вставить
###     3) Местонахождение площадки
###     4) Картографический код площадки
###     5) Тип площадки
###     6) Описание площадки
die "Использование: insertmegadata"
    ." <файл данных> <название площадки > <местонахождение> <код карты>
    <тип> <описание>\n"
    unless @ARGV == 6;

my $megalithFile    = $ARGV[0];
my $siteName        = $ARGV[1];
my $siteLocation    = $ARGV[2];
my $siteMapRef      = $ARGV[3];
my $siteType        = $ARGV[4];
my $siteDescription = $ARGV[5];

### Открытие файла данных для дополнения или аварийный выход
open MEGADATA, ">>$megalithFile"
    or die "Невозможно открыть $megalithFile для дополнения: $!\n";

### Создать новую запись
my $record = join( ":", $siteName, $siteLocation, $siteMapRef,
                        $siteType, $siteDescription );

### Вставить новую запись в файл
print MEGADATA "$record\n"
    or die "Ошибка записи в $megalithFile: $!\n";

### Закрыть файл данных по мегалитам
close MEGADATA
    or die "Ошибка закрытия $megalithFile: $!";

print "Вставлена запись по площадке: $siteName\n";

exit;
```

В этом примере просто открывается файл данных в режиме дополнения, и в открытый файл пишется новая запись. Как ни прост этот процесс, у него есть потенциальный недостаток. Эта одноуровневая база данных не обнаруживает вставки нескольких элементов данных с одинаковым ключом для поиска. Это значит, что если бы мы захотели вставить еще одну запись о Стоунхендже, то программа охотно позво-

лила бы нам это сделать, несмотря на то что запись для Стоунхенджа уже существует.

Это может составить проблему с точки зрения целостности данных. Прежде чем дописывать данные, стоило бы осуществить проверку того, что при этом не образуется дубликатов данных. Простым подходом будет объединение программы вставки и программы запроса данных.

Другим потенциальным (и более важным) недостатком является то, что система не сможет надежно обрабатывать те случаи, когда несколько пользователей пытаются добавить данные в базу. Поскольку такие проблемы возникают также при обновлении и удалении данных, мы более подробно обсудим эту тему в одном из последующих разделов этой главы.

Вставлять новые записи в файл с записями фиксированной длины просто. Вместо вывода каждого поля и символа-разделителя в описатель файла Perl для создания из данных записи фиксированной длины можно использовать функцию `pack()`.

Обновление данных

Обновление данных в плоских файлах немного сложнее. При запросе данных из базы мы просто последовательно просматривали базу данных, пока не находили нужную запись. Аналогично при вставке данных мы просто присоединяли новые данные, не имея представления о том, какие данные уже хранятся в базе.

Главная проблема при обновлении данных состоит в том, что должна быть возможность прочесть данные из файла, повозиться с ними некоторое время и затем отправить обратно в файл, не потеряв при этом записей.

Один из подходов состоит в том, чтобы загрузить в память всю базу, провести все обновления в копии базы, находящейся в памяти, а затем сбросить все обратно на диск. Второй подход заключается в том, чтобы поочередно читать из базы данных записи, делать в отдельной записи необходимые изменения и затем сразу вносить запись во временный файл. После того как обработаны все записи, можно заменить исходный файл данных временным файлом. Оба приема жизнеспособны, но мы предпочитаем второй по соображениям производительности: загрузка больших баз данных в память может быть очень ресурсоемкой.

В следующей короткой программе реализована вторая из этих стратегий, для того чтобы обновлять коды карт в базе данных с разделителями записей:

```
#!/usr/bin/perl -w
#
# ch02/updatemegadata/updatemegadata: Обновляет файл с данными о
#                                     мегалитах по указанной площадке.
```

```
#                                        Использует данные с разделителем-
#                                        двоеточием и обновляет поле
#                                         кода карты.

### Проверить, задал ли пользователь аргументы для поиска:
###     1) Имя файла, содержащего данные
###     2) Название искомой площадки
###     3) Новый код карты
die "Использование: updatemegadata <файл данных> <название площадки >
    <новый код карты>\n"
    unless @ARGV == 3;

my $megalithFile = $ARGV[0];
my $siteName     = $ARGV[1];
my $siteMapRef   = $ARGV[2];
my $tempFile     = "tmp.$$";

### Открытие файла данных для чтения или аварийный выход
open MEGADATA, "<$megalithFile"
    or die "Невозможно открыть $megalithFile: $!\n";

### Открыть временный файл с данными мегалитов на запись
open TMPMEGADATA, ">$tempFile"
    or die " Невозможно открыть временный файл $tempFile: $!\n";

### Просмотреть записи и найти нужную площадку
while ( <MEGADATA> ) {

    ### Быстрая предварительная проверка для максимальной скорости:
    ### Пропустить запись, если не то название площадки
    next unless m/^\Q$siteName:/;

    ### Разбить данные записи на отдельные поля
    ### (пусть $description содержит символ новой строки)
    my ( $name, $location, $mapref, $type, $description ) =
         split( /:/, $_ );

    ### Пропустить запись, если название площадки не соответствует.
    ### (Излишне ввиду проведенной выше проверки, но сохранено
    ### для совместимости с другими примерами.)
    next unless $siteName eq $name;

    ### Запись для обновления найдена, обновим значение кода карты
    $mapref = $siteMapRef;

    ### Соберем обновленную запись
    $_ = join( ":", $name, $location, $mapref, $type, $description );

}
continue {
```

```
### Запишем данные во временный файл
   print TMPMEGADATA $_
      or die "Ошибка записи в $tempFile: $!\n";
}

### Закроем входной файл с мегалитическими данными
close MEGADATA;

### Закроем временный выходной файл с мегалитическими данными
close TMPMEGADATA
   or die "Ошибка при закрытии $tempFile: $!\n";

### Теперь мы "зафиксируем" изменения, удалив старый файл...
unlink $megalithFile
   or die "Невозможно удалить прежний файл $megalithFile: $!\n";

### и переименовав новый, чтобы он заменил прежний.
rename $tempFile, $megalithFile
   or die "Невозможно переименовать '$tempFile' в '$megalithFile': $!\n";

exit 0;
```

Как видите, в этом примере мы поиграли в Perl мускулами, использовав для упрощения алгоритма цикл `while ... continue` и добавив для увеличения скорости предварительную проверку.

Эквивалентная программа, которая может работать с файлом записей фиксированной длины, очень похожа, но для изменения содержимого поля используется более быстрое обновление по месту. Принцип аналогичен описанному выше запросу с обработкой по месту: необязательно распаковывать и упаковывать все поля записи, достаточно просто обновить соответствующую часть каждой записи. Например:

```
### Просмотр всех записей в поисках нужной площадки
while ( <MEGADATA> ) {

    ### Предварительная проверка с целью ускорения:
    ### Пропустить запись, если в начале ее нет нужного названия
    next unless m/^\Q$siteName/;

    ### Пропустить запись, если извлеченное название площадки
    ### не соответствует требуемому
    next unless unpack( "x64 x64 A16", $_ ) eq $siteName;

    ### Осуществить замену по месту для обновления поля кода карты
    substr( $_, 64+64, 16, pack( "A16", $siteMapRef ) );

}
```

Такой прием действует быстрее, чем упаковка и распаковка каждой записи, хранимой в файле, поскольку выполняется минимальный

объем действий, необходимый для изменения соответствующих значений полей.

Можно заметить, что предварительная проверка в этом примере надежна не на 100 процентов, но этого и не требуется. Она должна перехватить большинство случаев несоответствия, чтобы оправдать себя в результате сокращения числа раз, когда выполняются более дорогостоящие операции распаковки и проверки значения поля. Можно согласиться, что это не самое убедительное применение идеи, но мы еще вернемся к ней в данной главе для более серьезного обсуждения.

Удаление данных

Последний вид действий, которые можно осуществлять над данными в одноуровневой базе, это удаление данных из базы. Мы будем обрабатывать файл по одной записи, пропуская данные через временный файл, как делали это при обновлении, а не заглатывать все данные в память и сбрасывать их на диск в конце.

При такой технологии запись в базе данных скорее опускается, чем фактически удаляется. Каждая запись считывается из файла, проверяется и записывается во временный файл. Когда встречается запись, которую нужно удалить, она просто *не* пишется во временный файл. В результате из базы данных удаляются всякие ее следы, хотя и довольно безыскусным способом.

Следующая программа может быть использована для удаления соответствующей записи из базы данных с разделителями при задании в качестве аргумента названия площадки, которую нужно удалить:

```
#!/usr/bin/perl -w
#
# ch02/deletemegadata/deletemegadata: Удаляет запись для заданной мега-
#                                     литической площадки. Разделителем
#                                     полей служит двоеточие.
#

### Проверить, задал ли пользователь аргументы:
###     1) Имя файла, содержащего данные
###     2) Название площадки, которую нужно удалить
die "Использование: deletemegadata <имя файла с данными> <название
    площадки>\n"
    unless @ARGV == 2;

my $megalithFile  = $ARGV[0];
my $siteName      = $ARGV[1];
my $tempFile      = "tmp.$$";

### Открытие файла с данными на чтение или аварийный выход
open MEGADATA, "<$megalithFile"
```

```
    or die "Невозможно открыть $megalithFile: $!\n";

### Открыть временный файл с мегалитическими данными на запись
open TMPMEGADATA, ">$tempFile"
    or die "Невозможно открыть временный файл $tempFile: $!\n";

### Просмотреть все записи в поиске нужной площадки
while ( <MEGADATA> ) {

    ### Извлечь из записи название площадки (первое поле)
    my ( $name ) = split( /:/, $_ );

    ### Сравнить название площадки с содержащимся в записи
    if ( $siteName eq $name ) {

        ### Если найдена запись, которую нужно удалить, пропустить ее
        ### и перейти к следующей записи
        next;
    }

    ### Внести исходную запись во временный файл
    print TMPMEGADATA $_
        or die "Ошибка записи $tempFile: $!\n";
    }

### Закроем входной файл с мегалитическими данными
close MEGADATA;

### Закроем временный выходной файл с мегалитическими данными
close TMPMEGADATA
    or die "Ошибка закрытия $tempFile: $!\n";

### Теперь "зафиксируем" изменения, удалив старый файл ...
unlink $megalithFile
    or die "Невозможно удалить старый $megalithFile: $!\n";

### и переименовав новый, чтобы он заменил прежний.
rename $tempFile, $megalithFile
     or die "Невозможно переименовать '$tempFile' в '$megalithFile': $!\
n";

exit 0;
```

Программный код для удаления записей из файла с записями фиксированной длины почти идентичен. Единственное изменение касается, как и можно ожидать, кода для извлечения значения поля:

```
### Извлечь из записи название площадки (первое поле)
my ( $name ) = unpack( "A64", $_ );
```

Как и обновление, удаление данных может стать источником проблем, если несколько пользователей пытается одновременно произвести в

данных изменения. Как бороться с этой проблемой, будет рассмотрено далее в этой главе.

Размещение в плоских файлах сложных данных

При обсуждении «плоских файлов» мы пока сохраняли, извлекали и обрабатывали данные только одного, но самого основного типа, – строкового. Как сохранять более сложные данные, такие как списки, хэши или данные большой степени вложенности, использующие ссылки?

Ответ состоит в преобразовании любых типов данных *в* строки. Технически это известно как *маршаллинг* (*marshalling*), или *сериализация* (*serializing*) данных. Список модулей Perl (Perl Module List)[1] содержит раздел, в котором перечислено несколько модулей Perl, осуществляющих пересылку данных.

Мы рассмотрим два самых популярных модуля: `Data::Dumper` и `Storable`, и увидим, как их можно использовать, для того чтобы внести некоторое оживление в наши плоские файлы. Эти приемы можно использовать и при хранении сложных структур данных Perl в реляционных базах с использованием DBI, поэтому будьте внимательны.

Модуль Perl Data::Dumper

Модуль `Data::Dumper` принимает список переменных Perl и записывает их значения *в виде кода Perl,* который при своем выполнении воссоздаст исходные значения, какими бы сложными они ни были.

Этот модуль позволяет легко и быстро сделать дамп состояния программы на Perl в читаемом виде. С его помощью можно также восстановить состояние программы, выполнив код дампа с помощью функции `eval()` или `do()`.

Проще всего показать, как это происходит, приведя несложный пример:

```
#!/usr/bin/perl -w
#
# ch02/marshal/datadumpertest: Создает несколько переменных Perl
#                              и выводит их в дамп. Затем мы сбрасываем
#                              значения переменных и применяем eval
#                              к переменным из дампа ...

use Data::Dumper;
```

[1] Список модулей Perl можно найти на *http://www.perl.com/CPAN/*.

```
### Настроим стиль вывода Data::Dumper
### Подробности в документации по Data::Dumper
if ($ARGV[0] eq 'flat') {
    $Data::Dumper::Indent = 0;
    $Data::Dumper::Useqq  = 1;
}
$Data::Dumper::Purity = 1;

### Создадим несколько переменных Perl
my $megalith  = 'Стоунхендж';
my $districts = [ 'Уилтшир', 'Оркни', 'Дорсет' ];

### Выведем их
print "Начальные значения: \$megalith  = " . $megalith . "\n" .
      "           \$districts = [ ". join(", ", @$districts) . " ]\n\n";

### Создадим из базы данных новый объект Data::Dumper
my $dumper = Data::Dumper->new( [    $megalith, $districts  ],
                                [ qw( megalith  districts ) ] );

### Выведем значения Perl в переменную
my $dumpedValues = $dumper->Dump();

### Покажем, во что Data::Dumper превратил переменные!
print "Код Perl, созданный Data::Dumper:\n";
print $dumpedValues . "\n";

### Присвоим переменным бессмысленные значения
$megalith = 'Blah! Blah!';
$districts = [ 'Alderaan', 'Mordor', 'The Moon' ];

### Выведем бессмысленные значения
print "Бессмысленные значения: \$megalith  = " . $megalith . "\n" .
      "           \$districts = [ ". join(", ", @$districts) . " ]\n\n";

### Загрузим переменные Perl с помощью eval
eval $dumpedValues;
die if $@;

### Выведем загруженные заново значения
print "Загруженные заново значения: \$megalith  = " . $megalith . "\n" .
      "           \$districts = [ ". join(", ", @$districts) . " ]\n\n";

exit;
```

В этом примере инициализируются две переменные Perl и выводятся их значения. Затем создается объект `Data::Dumper` **с этими значениями, изменяются исходные значения и выводятся новые, показывающие, что все без обмана. Наконец, к результату** `$dumper->Dump()` **применяется функция** `eval`**, в результате чего переменным возвращаются их исход-**

ные значения. И снова мы выводим их, чтобы показать, что никакой ловкости рук не используется:

```
Начальные значения: $megalith  = Стоунхендж
                    $districts = [ Уилтшир, Оркни, Дорсет ]

Код Perl, созданный Data::Dumper:
$megalith = 'Стоунхендж';
$districts = [
            'Уилтшир',
            'Оркни',
            'Дорсет'
          ];

Бессмысленные значения: $megalith  = Blah! Blah!
                        $districts = [ Alderaan, Mordor, The Moon ]
Загруженные заново значения: $megalith  = Стоунхендж
                             $districts = [Уилтшир, Оркни, Дорсет]
```

Как использовать `Data::Dumper` для оживления наших плоских файлов? Прежде всего, нужно попросить `Data::Dumper` создать плоскую выдачу, т. е. без символов новой строки. Для этого нужно установить две глобальные переменные в пакете:

```
$Data::Dumper::Indent = 0; # не использовать символы новой строки
                           # для структуризации выдачи
$Data::Dumper::Useqq  = 1; # заключать строки в двойные кавычки
                           # и использовать "\n" в конце строки
```

В нашей программе это можно сделать, передав ей в командной строке аргумент `flat`. В результате в соответствующем месте программа выдаст следующее:

```
$megalith = "Стоунхендж";$districts = ["Уилтшир","Оркни","Дорсет"];
```

Теперь можно модифицировать наши прежние сценарии просмотра (выборки), вставки, обновления и удаления так, чтобы использовать `Data::Dumper` для форматирования записей вместо функций `join()` или `pack()`, которые мы использовали прежде. Для распаковки записей вместо `split()` или `unpack()` будем использовать `eval`.

Вот главный цикл использовавшегося нами ранее сценария обновления (остальная его часть остается неизменной, за исключением добавления вверху строки `use Data::Dumper` и установки переменных `Data::Dumper`, как было описано выше):

```
### Просмотр всех записей в поисках нужной площадки
while ( <MEGADATA> ) {

    ### Быстрая предварительная проверка для максимальной скорости:
    ### Пропустить запись, если не то название площадки
    next unless m/\Q$siteName/;
```