Оглавление

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте **www.dmkpress.com**, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**; при этом напишите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в тексте, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты **dmkpress@gmail.com**.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Вступление

Системы баз данных широко распространены в корпоративном мире как видимый инструмент – сотрудники часто напрямую взаимодействуют с такими системами, чтобы отправить данные или создать отчеты. Но не менее часто они используются как невидимые компоненты программных систем. Например, представьте веб-сайт электронной коммерции, использующий базу данных на стороне сервера для хранения информации о клиентах, товарах и продажах. Или вообразите систему навигации GPS, использующую встроенную базу данных для управления картами дорог. В обоих этих примерах система баз данных скрыта от пользователя; с ней взаимодействует только код приложения.

С точки зрения разработчика программного обеспечения, обучение непосредственному использованию базы данных выглядит скучным занятием, потому что современные системы баз данных имеют интеллектуальные пользовательские интерфейсы, упрощающие создание запросов и отчетов. С другой стороны, включение поддержки базы данных в программное приложение выглядит более захватывающим, поскольку открывает множество новых и неисследованных возможностей.

Но что означает «включение поддержки базы данных»? Система баз данных обеспечивает множество новых возможностей, таких как долговременное хранение данных, поддержка транзакций и обработка запросов. Какие из этих возможностей необходимы, и как их интегрировать в программное обеспечение? Предположим, например, что программиста просят изменить существующее приложение и добавить возможность сохранения состояния, повысить надежность или эффективность доступа к файлам. Программист оказывается перед выбором между несколькими архитектурными вариантами. Он может:

- приобрести полноценную систему баз данных общего назначения, а затем изменить приложение и организовать в нем подключение к базе данных в качестве клиента;
- взять узкоспециализированную систему, реализующую только нужные функции, и встроить ее код непосредственно в приложение;
- написать необходимые функции самостоятельно.

Чтобы сделать правильный выбор, программист должен понимать последствия работы каждого из этих вариантов. Он должен знать не только то, что делают системы баз данных, но также как они это делают и почему.

В этой книге мы рассмотрим системы баз данных с точки зрения разработчика программного обеспечения. Это позволит нам понять, *почему* системы баз данных являются такими, какие они есть. Конечно, важно уметь писать запросы, но не менее важно знать, как они обрабатываются. Мы должны не просто уметь использовать JDBC, но и знать и понимать, почему API содержит именно

такие классы и методы. Мы должны понять, насколько сложно написать дисковый кеш или подсистему журналирования. И вообще, *что такое* драйвер базы данных.

Организация книги

Первые две главы содержат краткий обзор систем баз данных и принципов их использования. Глава 1 обсуждает назначение и особенности системы баз данных и знакомит с системами Derby и SimpleDB. Глава 2 рассказывает, как написать приложение базы данных на Java. В ней будут представлены основы JDBC – фундаментального API для программ на Java, взаимодействующих с базами данных.

В главах 3–11 рассматривается внутренние устройство типичного механизма базы данных. Каждая из этих глав охватывает отдельный компонент, начиная с самого низкого уровня абстракции (диспетчер дисков и файлов) и заканчивая интерфейсом самого верхнего уровня (интерфейс клиента JDBC). При обсуждении каждого компонента объясняются вероятные проблемы и рассматриваются возможные проектные решения. Благодаря такому подходу вы сможете увидеть, какие услуги предоставляет каждый компонент и как он взаимодействует друг с другом. На протяжении этой части книги вы будете наблюдать постепенное развитие простой, но вполне функциональной системы.

Остальные четыре главы посвящены эффективной обработке запросов. В них исследуются сложные приемы и алгоритмы, предназначенные для замены простых решений, описанных в предыдущей части. Среди всего прочего здесь рассматриваются: индексация, сортировка, интеллектуальная буферизация и оптимизация запросов.

Предварительные требования

Эта книга предназначена для студентов вузов старших курсов, изучающих курс информатики. Предполагается, что читатель знаком с основами программирования на Java, например он умеет использовать классы из `java.util`, в частности коллекции и ассоциативные массивы. Более сложные понятия из мира Java (такие как RMI и JDBC) будут полностью объяснены в тексте.

Сведения, представленные в данной книге, обычно изучаются в углубленном курсе по системам баз данных. Однако в моей преподавательской практике их с успехом усваивали студенты, не имеющие опыта работы с базами данных. Поэтому можно сказать, что для понимания идей, представленных в этой книге, не требуется иметь какие-либо знания о базах данных, кроме поверхностного знакомства с SQL. Впрочем, студенты, незнакомые с SQL, также смогут усвоить необходимые им знания.

Программное обеспечение SimpleDB

По моему опыту, студенты быстро схватывают концептуальные идеи (такие как управление параллелизмом и буферами или алгоритмы оптимизации запросов), но им сложнее понять, как эти идеи взаимодействуют. В идеале сту-

дент должен написать всю систему баз данных в своей курсовой работе, так же, как он написал бы целый компилятор при изучении курса по компиляторам. Однако системы баз данных намного сложнее компиляторов, поэтому такой подход выглядит непрактичным. Поэтому я решил сам написать простую, но полнофункциональную систему баз данных под названием *SimpleDB* и дать возможность студентам применить свои знания для изучения кода SimpleDB и его изменения.

По своим возможностям и структуре SimpleDB «выглядит» как коммерческая система баз данных. Функционально это многопользовательский сервер баз данных с поддержкой транзакций, который выполняет операторы SQL и взаимодействует с клиентами через интерфейс JDBC. Конструктивно она содержит те же основные компоненты, что и коммерческая система с похожим API. Каждому компоненту SimpleDB посвящена отдельная глава в книге, где обсуждается код компонента и проектные решения, стоящие за ним.

SimpleDB является прекрасным наглядным пособием благодаря небольшому объему кода, который легко читается и модифицируется. В ней отсутствует необязательная функциональность, реализована только малая часть языка SQL и используются только самые простые (и часто очень неоптимальные) алгоритмы. Как результат перед студентами открывается широкое поле возможностей расширения системы дополнительными функциями и применения более эффективных алгоритмов; многие из этих расширений предлагаются в конце каждой главы как упражнения.

Исходный код SimpleDB можно получить по адресу http://cs.bc.edu/~sciore/simpledb. Подробная информация об установке и использовании SimpleDB представлена на этой же веб-странице и в главе 1. Я с благодарностью приму любые предложения по улучшению кода, а также сообщения о любых ошибках. Пишите мне по адресу sciore@bc.edu.

Список дополнительных ресурсов в конце каждой главы

Главная задача этой книги – дать ответы на два основных вопроса: какие функции предоставляют системы баз данных и какие алгоритмы и проектные решения лучше всего подходят для их реализации? Книгами, посвященными различным аспектам этих вопросов, можно заполнить не одну книжную полку. Поскольку ни одна книга не может охватить все и вся, я решил представить только те алгоритмы и методы, которые наиболее четко иллюстрируют соответствующие проблемы. Моя главная цель – познакомить вас с основными принципами, даже если придется опустить (или сократить) обсуждение более жизнеспособной версии того или иного метода. Поэтому в конце каждой главы имеется раздел «Для дополнительного чтения». В этих разделах обсуждаются интересные идеи и направления исследований, не упомянутые в тексте, и даются ссылки на соответствующие веб-страницы, исследовательские статьи, справочные руководства и книги.

Упражнения в конце каждой главы

В конце каждой главы дается несколько упражнений. Некоторые, имеющие целью закрепить полученные знания, можно решить карандашом на бумаге. В других предлагаются интересные модификации в SimpleDB, и многие из них могут служить отличными программными проектами. Для большинства упражнений я написал свои решения. Если вы преподаете курс по этому учебнику и хотите получить копию руководства с решениями, напишите мне по адресу sciore@bc.edu.

Об авторе

Эдвард Сьоре (**Edward Sciore**) – недавно вышедший на пенсию доцент кафедры информатики в Бостонском колледже. Автор многочисленных статей о системах баз данных, охватывающих теорию и практику их разработки. Больше всего ему нравится преподавание курсов баз данных увлеченным студентам. Этот опыт преподавания, накопленный за 35-летний период, привел к написанию данной книги.

Глава 1

Системы баз данных

Системы баз данных играют важную роль в компьютерной индустрии. Некоторые из них (такие как Oracle) чрезвычайно сложны и, как правило, работают на больших высокопроизводительных компьютерах. Другие (такие как SQLite) имеют небольшой размер и предназначены для хранения данных отдельных приложений. Несмотря на широкий спектр применения, все системы баз данных имеют схожие черты. В этой главе рассматриваются проблемы, которые должна решать система баз данных, и возможности, которыми она должна обладать. Здесь также будут представлены системы баз данных Derby и SimpleDB, обсуждающиеся далее в этой книге.

1.1. Зачем нужны системы баз данных?

База данных – это коллекция данных, хранящаяся на компьютере. Обычно информация в базе данных организована в *записи*, например в записи с информацией о сотрудниках, медицинские записи, записи о продажах и т. д. В табл. 1.1 представлена база данных, хранящая информацию о студентах университета и изученных ими курсах. Эта база данных будет использоваться как учебный пример на протяжении всей книги. В базе данных в табл..1 хранятся записи пяти типов:

- для каждого студента, учившегося в университете, имеется запись STUDENT; каждая запись содержит идентификационный номер студента, имя, год выпуска и идентификационный номер основной кафедры;
- для каждой кафедры в университете имеется запись DEPT; каждая запись содержит идентификационный номер кафедры и название;
- для каждого обучающего курса, предлагаемого университетом, имеется запись COURSE; каждая запись содержит идентификационный номер курса, название и идентификационный номер кафедры, которая его предлагает;
- для каждого раздела курса, который когда-либо читался, имеется запись SECTION; каждая запись содержит идентификационный номер раздела, год, когда был предложен этот раздел, идентификационный номер курса и имя преподавателя, читающего этот раздел;
- для каждого курса, который прослушал студент, имеется запись ENROLL; каждая запись содержит идентификационные номера зачисления, студента и раздела пройденного курса, а также оценку, полученную студентом за курс.

Таблица 1.1. Примеры записей в университетской базе данных

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2021	10
	2	amy	2020	20
	3	max	2022	10
	4	sue	2022	20
	5	bob	2020	30
	6	kim	2020	20
	7	art	2021	30
	8	pat	2019	20
	9	lee	2021	10

DEPT	DId	DName
	10	compsci
	20	math
	30	drama

COURSE	CId	Title	DeptId
	12	db	systems
	22	compilers	10
	32	calculus	20
	42	algebra	20
	52	acting	30
	62	elocution	30

ENROLL	EId	StudentId	SectionId	Grade
	14	1	13	A
	24	1	43	C
	34	2	43	B+
	44	4	33	B
	54	4	53	A
	64	6	53	A

SECTION	SectId	CourseId	Prof	YearOffered
	13	12	turing	2018
	23	12	turing	2016
	33	32	newton	2017
	43	32	einstein	2018
	53	62	brando	2017

Таблица 1.1 – это всего лишь концептуальная схема. Она ни в коей мере не отражает порядок хранения записей или особенности доступа к ним. Существует множество программных продуктов, называемых *системами баз данных*, которые предоставляют обширный набор средств управления записями.

Что имеется в виду под «управлением» записями? Какие возможности должны иметься в системе баз данных, а какие могут отсутствовать? Вот пять основных требований:

- *базы данных должны обеспечивать долговременное хранение*; в противном случае записи исчезнут после выключения компьютера;
- *базы данных могут быть общими*; многие базы данных, такие как наша университетская база данных, предназначены для одновременного использования несколькими пользователями;
- *базы данных должны гарантировать* безошибочное хранение информации; если пользователи не смогут доверять содержимому базы данных, она станет бесполезной;
- *базы данных могут быть очень большими*; база данных в табл. 1.1 содержит всего 29 записей, что смехотворно мало – базы данных с миллионами (и даже миллиардами) записей совсем не редкость;
- *базы данных должны быть удобными*; если пользователи не смогут с легкостью получать нужные данные, их производительность снизится, и они будут требовать использовать другой продукт.

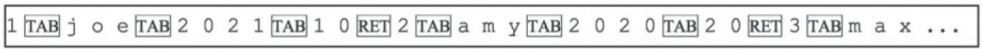

Рис. 1.1. Реализация записи STUDENT в текстовом файле

В следующих подразделах рассматриваются следствия этих требований. Каждое требование вынуждает включать в систему баз данных все больше и больше функций, из-за чего она оказывается намного сложнее, чем можно было бы ожидать.

1.1.1. Хранилище записей

Типичный способ обеспечить сохранность данных – хранить записи в файлах. А самый простой подход к хранению записей в файлах – сохранение их в текстовых файлах, по одному файлу для каждого типа записей; каждую запись можно представить в виде текстовой строки, в которой значения отделены друг от друга символами табуляции. На рис. 1.1 показано начало текстового файла с записями типа STUDENT.

Преимущество этого подхода в том, что пользователь может просматривать и изменять файлы с помощью простого текстового редактора. К сожалению, этот подход слишком неэффективен по двум причинам.

Первая причина в том, что для изменения данных в больших текстовых файлах требуется слишком много времени. Представьте, например, что кто-то решил удалить из файла STUDENT запись с информацией о студенте Joe. У системы баз данных не будет иного выбора, кроме как переписать файл, начав с записи о студенте Amy. Короткий файл будет переписан быстро, но на перезапись файла объемом 1 Гбайт легко может уйти несколько минут, а это недопустимо долго. Система баз данных должна использовать более оптимальные способы хранения записей, чтобы при изменении данных достаточно было перезаписать локальные фрагменты.

Вторая причина – для чтения больших текстовых файлов требуется слишком много времени. Представьте поиск в файле STUDENT всех студентов, выпущенных в 2019 году. Единственный способ – просканировать файл от начала до конца. Последовательное сканирование может быть очень неэффективным. Возможно, вам знакомы некоторые структуры данных, такие как деревья и хеш-таблицы, которые обеспечивают быстрый поиск. Система баз данных должна использовать аналогичные структуры для реализации своих файлов. Например, система баз данных может организовать записи в файле, используя структуру, ускоряющую один конкретный тип поиска (например, по имени студента, году выпуска или специальности), или создать несколько вспомогательных файлов, каждый из которых ускоряет свой тип поиска. Такие вспомогательные файлы называются *индексами* и являются предметом обсуждения главы 12.

1.1.2. Многопользовательский доступ

Когда базой данных пользуется несколько человек, есть вероятность, что они одновременно обратятся к одним и тем же ее файлам. Параллелизм – хорошая штука, потому что избавляет пользователей от необходимости ждать, пока другие закончат работу. Но параллелизм может привести к появлению ошибок в базе

данных. Например, представьте базу данных в системе бронирования билетов на авиарейсы. Предположим, что два клиента пытаются забронировать билеты на рейс, в котором осталось 40 мест. Если оба пользователя одновременно прочитают одну и ту же запись, они оба увидят 40 доступных мест. Затем они оба изменят эту запись так, чтобы в ней осталось 39 свободных мест. В результате такого резервирования двух мест в базе данных будет зарегистрировано только одно место.

Решением этой проблемы является ограничение параллелизма. Система баз данных должна позволить первому пользователю прочитать запись о рейсе и увидеть 40 доступных мест, но заблокировать второго пользователя до того момента, пока первый не закончит работу. Когда второй пользователь получит возможность продолжить работу, он увидит 39 доступных мест и уменьшит это количество до 38, как и должно быть. То есть система баз данных должна обнаруживать ситуации, когда один пользователь собирается выполнить действие, конфликтующее с действиями другого пользователя, и тогда (и только тогда) блокировать работу этого пользователя, пока первый не завершит операцию.

Пользователям также может понадобиться отменить свои изменения. Например, представьте, что пользователь выполнил поиск в базе данных бронирования билетов и нашел дату, на которую есть свободные места на рейс в Мадрид. Затем он нашел свободный номер в гостинице на ту же дату. Но пока пользователь бронировал место на рейс, все номера в гостиницах на эту дату были забронированы другими людьми. В этом случае пользователю может потребоваться отменить бронирование рейса и попробовать выбрать другую дату.

Изменение, которое еще можно отменить, не должно быть видно другим пользователям базы данных. В противном случае другой пользователь, увидев изменение, может подумать, что данные «настоящие», и принять решение на их основе. Поэтому система баз данных должна давать пользователям возможность указывать, когда их изменения можно сохранить, или, как говорят, *зафиксировать* (подтвердить). Как только пользователь зафиксирует изменения, они становятся видимыми всем остальным и уже не могут быть отменены. Этот вопрос рассматривается в главе 5.

1.1.3. Обработка аварийных ситуаций

Представьте, что вы запустили программу, которая повышает зарплату всем профессорам, и вдруг система баз данных аварийно завершается. После перезапуска системы вы понимаете, что кому-то из профессоров уже назначена новая зарплата, а кому-то еще нет. Как быть в такой ситуации? Нельзя просто повторно запустить программу, потому что кто-то из профессоров получит двойную прибавку. То есть вам нужно, чтобы система баз данных правильно восстановилась после сбоя, отменив все изменения, вносившиеся в момент сбоя. Для этого используется интересный и нетривиальный механизм, который рассматривается в главе 5.

1.1.4. Управление памятью

Базы данных должны хранить информацию в устройствах долговременной памяти, таких как жесткие диски или твердотельные накопители на основе флеш-памяти. Твердотельные накопители действуют примерно в 100 раз быстрее жестких дис-

ков, но и стоят значительно дороже. Типичное время доступа к диску составляет примерно 6 мс, а к флеш-памяти – 60 мкс. Однако оба этих устройства на несколько порядков медленнее оперативной памяти (или ОЗУ), время доступа которой составляет около 60 нс. То есть операции с ОЗУ выполняются примерно в 1000 раз быстрее, чем с флеш-памятью, и в 100 000 раз быстрее, чем с жестким диском.

Чтобы почувствовать разницу в производительности и увидеть, какие проблемы она порождает, рассмотрим следующую аналогию. Предположим, вы очень хотите шоколадное печенье. Есть три способа получить его: взять у себя на кухне, сходить в соседний продуктовый магазин или заказать доставку по почте. В этой аналогии кухня соответствует оперативной памяти, соседний магазин – флеш-накопителю, заказ по почте – диску. Допустим, чтобы взять печенье на кухне, требуется всего 5 секунд. Поход в магазин займет 5000 секунд (это больше часа): вам понадобится добраться до магазина, выстоять длинную очередь, купить печенье и вернуться домой. А для получения печенья по почте потребуется 500 000 секунд – больше 5 дней: вы должны будете заказать печенье через интернет, а компания отправит его вам обычной почтой. С этой точки зрения флеш-память и диски выглядят очень медленными.

Но это еще не все! Механизмы поддержки параллелизма и обеспечения надежности замедляют работу еще больше. Если данные, которые вам нужны, уже использует кто-то еще, вам придется подождать, пока эти данные будут освобождены. В нашей аналогии это можно представить так: вы пришли в магазин и обнаружили, что печенье распродано. В этом случае вам придется подождать, пока завезут новую партию.

Другими словами, система баз данных должна соответствовать противоречивым требованиям: управлять бóльшим объемом данных, чем умещается в оперативную память, используя медленные устройства, обслуживать одновременно множество людей, конкурирующих за доступ к данным, и обеспечить возможность полного восстановления этих данных, сохраняя при этом разумное время отклика.

Большая часть решения этой головоломки заключается в использовании кеширования. Всякий раз, когда требуется обработать запись, система баз данных загружает ее в оперативную память и старается хранить ее там как можно дольше. При таком подходе часть базы данных, используемая в данный момент, будет находиться в оперативной памяти. Все операции чтения и записи будут выполняться с оперативной памятью. Эта стратегия позволяет вместо медленной долговременной памяти использовать быструю оперативную память, но она имеет существенный недостаток – версия базы данных на устройствах хранения может устареть. В системе баз данных должны быть реализованы методы надежной синхронизации версии базы данных на устройствах хранения с версией в ОЗУ, даже в случае сбоя (когда содержимое ОЗУ уничтожается). Различные стратегии кеширования мы рассмотрим в главе 4.

1.1.5. Удобство

База данных может оказаться бесполезной, если пользователи не смогут с легкостью извлекать нужные данные. Например, представьте, что пользователь решил узнать имена всех студентов, окончивших учебу в 2019 году. В отсутствие системы баз данных пользователь будет вынужден написать програм-

му для сканирования файла со списком студентов. В листинге 1.1 показан код на Java такой программы, в котором предполагается, что файл хранит записи в текстовом виде. Обратите внимание, что большая часть кода связана с декодированием файла, чтением каждой записи и преобразованием ее в массив значений. Код, сопоставляющий фактические значения с искомыми (выделен жирным шрифтом) и извлекающий имена студентов, скрыт внутри малоинтересного кода, манипулирующего файлом.

Листинг 1.1. Поиск имен студентов, закончивших обучение в 2019 году

```
public static List<String> getStudents2019() {
    List<String> result = new ArrayList<>();
    FileReader rdr = new FileReader("students.txt");
    BufferedReader br = new BufferedReader(rdr);
    String line = br.readLine();
    while (line != null) {
        String[] vals = line.split("\t");
        String gradyear = vals[2];
        if (gradyear.equals("2019"))
            result.add(vals[1]);
        line = br.readLine();
    }
    return result;
}
```

По этой причине большинство систем баз данных поддерживают язык запросов, чтобы пользователи могли легко определить искомые данные. Стандартным языком запросов для реляционных баз данных является SQL. Код в листинге 1.1 можно выразить одним оператором SQL:

```
select SName from STUDENT where GradYear = 2019
```

Этот оператор намного короче и нагляднее, чем программа на Java, в первую очередь потому, что определяет, какие значения требуется извлечь из файла, а не как они должны извлекаться.

1.2. Система баз данных Derby

Изучение идей, лежащих в основе баз данных, протекает намного эффективнее, если есть возможность наблюдать за работой системы баз данных в интерактивном режиме. В настоящее время существует множество доступных систем баз данных, но я предлагаю использовать *Derby*, потому что она написана на Java, распространяется бесплатно, проста в установке и в использовании. Последнюю версию Derby можно загрузить на вкладке downloads, на странице db.apache.org/derby. Загруженный файл дистрибутива нужно распаковать в папку для установки, после чего в ней вы сможете обнаружить несколько каталогов. Например, каталог `docs` содержит справочную документацию, каталог `demo` – образец базы данных и т. д. Эта система обладает гораздо более широким кругом возможностей, чем можно описать здесь, поэтому те, кому это интересно, могут прочитать различные руководства в каталоге `docs`.

Derby имеет множество функций, которые не затрагиваются в этой книге. На самом деле вам нужно добавить в путь поиска классов `classpath` четыре фай-

ла из каталога lib: derby.jar, derbynet.jar, derbyclient.jar и derbytools.jar. Сделать это можно множеством способов, в зависимости от платформы Java и операционной системы. Я покажу, как это сделать, на примере платформы разработки Eclipse. Если вы незнакомы с Eclipse, то можете скачать ее код и документацию с сайта eclipse.org. Использующие другие платформы разработки смогут адаптировать мои указания для Eclipse под особенности своего окружения.

Прежде всего создайте в Eclipse проект для сборки Derby. Затем настройте путь сборки: в диалоге Properties (Свойства) выберите слева пункт Java Build Path (Путь сборки Java); щелкните на вкладке Libraries (Библиотеки), затем на кнопке Add External JARS (Добавить внешние JAR) и в открывшемся диалоге выберите четыре JAR-файла, перечисленных выше. Вот и все!

Дистрибутив Derby содержит приложение ij, помогающее создавать и использовать базы данных Derby. Так как Derby целиком написана на Java, название ij фактически является именем класса Java, находящегося в пакете org.apache.derby.tools. Запустить ij можно, выполнив этот класс. Чтобы выполнить класс из Eclipse, откройте диалог Run Configurations (Запуск конфигурации) в меню Run (Выполнить). Добавьте новую конфигурацию в свой проект Derby; дайте ей имя «Derby ij». В поле, определяющее главный класс, введите «org.apache.derby.tools.ij». После запуска этой конфигурации ij отобразит окно консоли с приглашением к вводу.

В данной консоли можно вводить последовательности команд. Команда – это строка, заканчивающаяся точкой с запятой. Команды можно вводить в несколько строк; клиент ij не выполнит команду, пока не встретит строку, заканчивающуюся точкой с запятой. Любой оператор SQL считается допустимой командой. Кроме того, ij поддерживает команды подключения и отключения от базы данных и завершения сеанса.

Команда connect должна иметь аргумент, определяющий базу данных, и подключается к ней, а команда disconnect отключается от нее. В одном сеансе можно подключаться к базе данных и отключаться от нее сколько угодно раз. Команда exit завершает сеанс. В листинге 1.2 показан пример сеанса ij. Сеанс состоит из двух частей. В первой части пользователь подключается к новой базе данных, создает таблицу, добавляет в нее запись и отключается. Во второй части пользователь повторно подключается к этой же базе данных, извлекает добавленные значения и отключается.

Листинг 1.2. Пример сеанса ij

```
ij> connect 'jdbc:derby:ijtest;create=true';
ij> create table T(A int, B varchar(9));
0 rows inserted/updated/deleted
ij> insert into T(A,B) values(3, 'record3');
1 row inserted/updated/deleted
ij> disconnect;
ij> connect 'jdbc:derby:ijtest';
ij> select * from T;
A          |B
----------------------
3          |record3
1 row selected
ij> disconnect;
ij> exit;
```

Аргумент команды `connect` называется *строкой подключения*. Строка подключения состоит из трех компонентов, разделенных двоеточиями. Первые два компонента – «jdbc» и «derby» – сообщают, что команда должна подключиться к базе данных Derby с использованием протокола JDBC (обсуждается в главе 2). Третий компонент идентифицирует базу данных. В данном случае «ijtest» – это имя базы данных; ее файлы будут находиться в папке с именем «ijtest», расположенной в каталоге, откуда было запущено приложение `ij`. Например, если запустить программу из Eclipse, папка базы данных окажется в каталоге проекта. Подстрока «create = true» сообщает системе Derby, что та должна создать новую базу данных; если опустить эту подстроку (как во второй команде `connect`), тогда Derby будет пытаться открыть существующую базу данных.

1.3. Механизмы баз данных

Приложение баз данных, такое как `ij`, состоит из двух независимых частей: пользовательского интерфейса и кода для доступа к базе данных. Этот код называется *механизмом (или движком) базы данных*. Отделение пользовательского интерфейса от движка базы данных – типичный архитектурный прием, упрощающий разработку приложений. Хорошо известным примером такого разделения может служить система баз данных Microsoft Access. Она имеет графический пользовательский интерфейс, позволяющий пользователю взаимодействовать с базой данных, щелкая мышью и вводя значения, а движок выполняет операции с хранилищем данных. Когда пользовательский интерфейс определяет, что ему нужна информация из базы данных, он создает запрос и передает его движку. После этого движок выполняет запрос и передает полученные значения обратно в пользовательский интерфейс.

Такое разделение также добавляет гибкости системе: разработчик приложения может использовать один и тот же пользовательский интерфейс с разными движками баз данных или создавать разные пользовательские интерфейсы для одного и того же движка. Microsoft Access как раз является примером каждого случая. Форма, созданная в пользовательском интерфейсе Access, может подключаться к движку Access или к любому другому механизму баз данных. Ячейки в электронной таблице Excel могут содержать формулы, генерирующие запросы к движку Access.

Пользовательский интерфейс обращается к базе данных, подключаясь к нужному движку и вызывая его методы. Например, обратите внимание, что программа `ij` на самом деле является простым пользовательским интерфейсом. Ее команда `connect` устанавливает соединение с указанным движком базы данных, а каждая команда посылает оператор SQL движку, получает результаты и отображает их.

Механизмы баз данных обычно поддерживают несколько стандартных API. Когда Java-программа подключается к движку, она выбирает API, который называется JDBC. Глава 2 подробно обсуждает JDBC и показывает, как написать `ij`-подобное приложение с использованием JDBC.

Пользовательский интерфейс может подключаться к *встроенному* механизму базы данных или *серверному*. Когда используется встроенный механизм базы данных, движок действует в том же процессе, что и код пользовательско-

го интерфейса. Это дает пользовательскому интерфейсу эксклюзивный доступ к движку. Встроенный движок должен использоваться, только когда база данных «принадлежит» данному приложению и хранится на том же компьютере, где выполняется приложение. В иных случаях приложения должны использовать серверные движки.

Когда используется соединение с сервером, код движка базы данных выполняется внутри специальной серверной программы. Эта серверная программа работает постоянно, ожидая запросов на соединение от клиентов, и не обязательно должна находиться на том же компьютере, что и ее клиенты. Когда клиент установит соединение с сервером, он посылает ему JDBC-запросы и получает ответы.

К серверу могут подключиться сразу несколько клиентов. Пока сервер обрабатывает запрос одного клиента, другие клиенты могут посылать свои запросы. В серверной программе имеется *планировщик*, который ставит запросы в очередь ожидания обслуживания и определяет, когда они будут обработаны. Никто из клиентов не знает о существовании других клиентов, и каждый из них полагает (если не учитывать задержки, вызванные работой планировщика), что сервер имеет дело исключительно с ним.

Сеанс `ij` в листинге 1.2 подключается ко встроенному движку. Он создал базу данных «ijtest» на том же компьютере, где выполняется, не обращаясь ни к какому серверу. Чтобы выполнить аналогичные действия на сервере, необходимо следующее: запустить движок Derby как сервер и изменить команду `connect` так, чтобы она ссылалась на сервер.

Код серверного движка Derby находится в Java-классе `NetworkServerControl`, в пакете `org.apache.derby.drda`. Чтобы запустить сервер из Eclipse, откройте диалог Run Configurations (Запуск конфигурации) в меню Run (Выполнить). Добавьте в проект Derby новую конфигурацию с именем «Derby Server». В поле с именем основного класса введите «org.apache.derby.drda.NetworkServerControl». На вкладке Arguments (Аргументы) введите аргумент «start -h localhost». После запуска конфигурации должно появиться окно консоли, сообщающее, что сервер Derby запущен.

Что означает аргумент «start -h localhost»? Первое слово – это команда «start», сообщающая классу, что тот должен запустить сервер. Остановить сервер можно, выполнив тот же класс с аргументом «shutdown» (или просто завершив процесс в окне консоли). Строка «-h localhost» указывает, что сервер должен принимать запросы только от клиентов, действующих на одном с ним компьютере. Если заменить «localhost» доменным именем или IP-адресом, сервер будет принимать запросы только от компьютера с этим именем или IP-адресом. Если указать IP-адрес «0.0.0.0», сервер будет принимать запросы от любых компьютеров[1].

Строка подключения к серверу должна определять сеть или IP-адрес сервера. Например, взгляните на следующие команды `connect`:

---

[1] Имейте в виду, что, разрешая подключаться к серверу любым клиентам, вы открываете базу данных не только для законопослушных пользователей, но и для злоумышленников. Обычно подобные серверы размещают за брандмауэром или включают механизм аутентификации Derby или и то, и другое.

```
ij> connect 'jdbc:derby:ijtest'
ij> connect 'jdbc:derby://localhost/ijtest'
ij> connect 'jdbc:derby://cs.bc.edu/ijtest'
```

Первая команда подключается ко встроенному движку базы данных «ijtest». Вторая устанавливает соединение с серверным движком базы данных «ijtest», действующим на компьютере «localhost», то есть на локальной машине. Третья команда устанавливает соединение с серверным движком базы данных «ijtest», действующим на удаленном компьютере «cs.bc.edu».

Обратите внимание, что строка подключения однозначно определяет выбор встроенного или серверного движка. Например, вернемся к листингу 1.2. Мы можем изменить сеанс и использовать подключение к серверу, просто изменив строку подключения в команде `connect`. Другие команды в сеансе не требуют изменений.

1.4. Система баз данных SimpleDB

Derby – это сложная, полноценная система баз данных, поэтому ее исходный код трудно понять или изменить. В противовес Derby я написал систему баз данных SimpleDB. Ее реализация уместилась в небольшой объем кода, который легко читается и легко модифицируется. В ней отсутствует вся необязательная функциональность, реализовано ограниченное подмножество языка SQL и используются только самые простые (и часто неоптимальные) алгоритмы. Ее цель – помочь вам получить четкое представление о каждом компоненте механизма базы данных и о взаимодействиях между ними.

Последнюю версию SimpleDB можно загрузить с веб-сайта cs.bc.edu/~sciore/simpledb. Загруженный файл архива следует распаковать в папку `SimpleDB_3.x`, после этого в ней появятся каталоги `simpledb`, `simpleclient` и `derbyclient`. В папке `simpledb` находится код механизма базы данных. В отличие от Derby, этот код не упакован в архив JAR, поэтому все файлы находятся непосредственно в папке.

Чтобы установить движок SimpleDB, добавьте папку `simpledb` в путь поиска классов classpath. Для этого в Eclipse создайте новый проект с именем «SimpleDB Engine», затем скопируйте содержимое каталога `simpledb` из папки `SimpleDB_3.x` в каталог `src` проекта и обновите проект в Eclipse, выбрав пункт Refresh (Обновить) в меню File (Файл).

В папке `derbyclient` находятся примеры программ, использующих движок Derby. Скопируйте содержимое этой папки (не саму папку) в папку `src` проекта Derby и обновите его. Эти программы мы будем рассматривать в главе 2.

В папке `simpleclient` находятся примеры программ, использующих движок SimpleDB. Для экспериментов с ними создайте новый проект с именем «SimpleDB Clients». Чтобы эти программы смогли найти код движка SimpleDB, добавьте проект «Engine SimpleDB» в путь сборки проекта «SimpleDB Clients». Затем скопируйте содержимое папки `simpleclient` в каталог `src` проекта «SimpleDB Clients».

SimpleDB поддерживает оба типа движков – встроенный и серверный. В папке `simpleclient` можно найти программу `SimpleIJ`, которая является упрощенной версией программы `ij` из Derby. Одно из отличий от `ij` состоит в том, что `SimpleIJ` позволяет подключиться к движку только один раз, в начале сеанса. После запуска программа предложит вам ввести строку подключения. Синтаксис строки подключения аналогичен синтаксису в `ij`, например:

```
jdbc:simpledb:testij
jdbc:simpledb://localhost
jdbc:simpledb://cs.bc.edu
```

Первая строка подключения описывает подключение к базе данных «testij» через встроенный движок. Так же как при использовании Derby, база данных должна находиться в каталоге программы – в данном случае клиента из проекта «SimpleDB Clients». В отличие от Derby, SimpleDB в любом случае создаст базу данных, если она отсутствует, поэтому нет необходимости добавлять флаг «create = true».

Вторая и третья строки подключения ссылаются на удаленную базу данных и используют для подключения серверный движок SimpleDB, действующий на локальной машине или на сервере `cs.bc.edu`. В отличие от Derby, эти строки подключения не описывают базу данных. Причина в том, что движок SimpleDB может обслуживать только одну базу данных, которая была определена в момент его запуска.

После выполнения каждой команды программа `SimpleIJ` снова выводит приглашение к вводу, предлагая ввести следующую команду. В отличие от Derby, вся команда целиком должна находиться в одной строке и завершаться точкой с запятой. После ввода команды программа выполнит ее. Если команда является запросом, в ответ выводится таблица с результатами. Если команда произвела какие-то изменения в базе данных, тогда выводится количество затронутых записей. Если команда содержит ошибку, будет выведено сообщение об ошибке. Движок SimpleDB поддерживает очень ограниченное подмножество языка SQL и генерирует исключение, если ему не удалось распознать команду. Эти ограничения описаны в следующем разделе.

Движок SimpleDB может действовать в режиме сервера. Основным классом в этом случае является `StartServer` из пакета `simpledb.server`. Чтобы запустить сервер из Eclipse, откройте диалог Run Configurations (Запуск конфигурации) в меню Run (Выполнить). Добавьте новую конфигурацию в проект «SimpleDB Engine» с названием «SimpleDB Server». В поле, определяющее главный класс, введите «simpledb.server.StartServer». На вкладке Arguments (Аргументы) введите имя базы данных. По умолчанию, в отсутствие аргумента, сервер использует базу данных «studentdb». После запуска конфигурации должно появиться окно консоли, сообщающее, что сервер SimpleDB запущен.

Сервер SimpleDB принимает соединения от любых клиентов, по аналогии с сервером Derby, запущенным с ключом «-h 0.0.0.0». Остановить сервер можно, только принудительно прервав его работу в консоли.

1.5. Версия SQL, поддерживаемая в SimpleDB

Движок Derby поддерживает почти весь стандартный набор операторов SQL. В отличие от Derby, SimpleDB реализует лишь небольшую часть стандарта и накладывает некоторые дополнительные ограничения. В этом разделе я лишь кратко перечислю эти ограничения. Более подробно они будут описаны в других главах книги, а в отдельных упражнениях, что приводятся в конце каждой главы, вам будет предложено реализовать некоторые недостающие функции.

Запросы в SimpleDB могут включать только предложения *select-from-where*, где `select` содержит список имен полей (без ключевого слова `AS`), а предложение `from` – список имен таблиц (без переменных области значений).

Выражения в необязательном предложении `where` можно объединять только с помощью логического оператора `and`. Выражения могут проверять лишь равенство полей константам. Движок SimpleDB не поддерживает другие стандартные операторы сравнения, логические и арифметические операторы и встроенные функции, а также скобки. Как следствие не поддерживаются вложенные запросы, агрегирование и вычисляемые значения.

Поскольку переменные области значений и псевдонимы полей не поддерживаются, все имена полей в запросе должны быть уникальными. А так как не поддерживаются предложения `group by` и `order by`, в запросах нельзя организовать группировку и сортировку. Вот еще некоторые ограничения:

- сокращенная форма «*» определения списка полей в предложении `select` не поддерживается;
- не поддерживаются неопределенные (NULL) значения;
- не поддерживаются явные и внешние соединения в предложении `from`;
- не поддерживается ключевое слово `union`;
- инструкция `insert` принимает только явные значения, то есть вставляемое значение нельзя определить с использованием вложенного запроса;
- инструкция `update` принимает лишь один оператор присваивания в предложении `set`.

1.6. Итоги

- *База данных* – это коллекция данных, хранящаяся на компьютере. Данные в базе обычно организованы в *записи*. *Система баз данных* – это программное обеспечение, управляющее записями в базе данных.
- Система баз данных должна быть способна обрабатывать большие базы данных, хранящиеся в медленной долговременной памяти, а также предоставлять высокоуровневый интерфейс для доступа к данным и *гарантировать* безошибочное хранение информации, разрешая конфликты между изменениями, вносимыми пользователями, и восстанавливая после сбоев системы. Системы баз данных отвечают этим требованиям благодаря наличию следующих функций:
  - хранение записей в файлах с использованием форматов, обеспечивающих более высокую эффективность доступа, чем позволяют стандартные средства файловой системы;
  - сложные алгоритмы индексирования для поддержки быстрого доступа;
  - возможность одновременно обслуживать нескольких пользователей с блокированием операций при необходимости;
  - поддержка фиксации и отката изменений;
  - кеширование записей из базы данных в оперативной памяти и управление синхронизацией между постоянной и оперативной версиями базы данных, с возможностью восстановления данных до состояния, предшествовавшего сбою;

- компилятор/интерпретатор языка для преобразования пользовательских запросов в выполняемый код;
- поддержка стратегий оптимизации запросов для преобразования неэффективных запросов в более эффективные.

- *Механизм (движок) базы данных* – это компонент системы баз данных, обеспечивающий доступ к данным и их хранение. Приложение базы принимает ввод пользователя и выводит результаты, для получения которых вызывает движок базы данных.
- Движок базы данных может быть встроенным в приложение или действовать как сервер. Программа со встроенным движком имеет эксклюзивный доступ к базе данных. Программа, соединяющаяся с сервером, использует движок вместе с другими программами, выполняющимися параллельно.
- *Derby* и *SimpleDB* – это две системы баз данных, написанные на Java. Derby реализует весь стандарт SQL, а SimpleDB – только ограниченное подмножество. SimpleDB можно использовать как учебный пример, благодаря простому и понятному коду. В остальной части книги, начиная с главы 3, мы последовательно исследуем этот код.

1.7. Для дополнительного чтения

Системы баз данных претерпели кардинальные изменения за эти годы. Подробный перечень этих изменений можно найти в главе 6 National Research Council (1999) и Haigh (2006). Желающие также могут прочитать статью в Википедии: en.wikipedia.org/wiki/Database_management_system#History[1].

Парадигма клиент–сервер с успехом используется во многих областях, а не только в базах данных. Общий обзор можно найти в Orfali et al. (1999). Описание функций и параметров конфигурации сервера Derby можно найти по адресу: db.apache.org/derby/manuals/index.html.

Haigh, T. (2006). «A veritable bucket of facts». Origins of the data base management system. ACM SIGMOD Record, 35(2), 33–49.

National Research Council Committee on Innovations in Computing and Communications. (1999). «Funding a revolution». National Academy Press. Доступен по адресу: www.nap.edu/read/6323/chapter/8#159.

Orfali, R., Harkey, D., & Edwards, J. (1999). «Client/server survival guide (3rd ed.)». Wiley.

1.8. Упражнения

Теория

1.1. Представьте, что в вашей организации возникла потребность управлять относительно небольшим набором записей (например, 100 или около того) и обеспечить общий доступ к ним.

а) Имеет ли смысл использовать для этого коммерческую систему баз данных?

[1] ru.wikipedia.org/wiki/База_данных#История. – *Прим. перев.*

b) Какие возможности системы баз данных могут остаться невостребованными?
c) Разумно ли использовать электронную таблицу для хранения этих записей? Какие проблемы могут при этом возникнуть?

1.2. Представьте, вам потребовалось организовать хранение большого объема личных данных в базе. Какие возможности системы баз данных вам не понадобятся?

1.3. Представьте, что у вас есть некоторые данные, для хранения которых вы не используете систему баз данных (например, список покупок, адресная книга, сведения о вкладах в банке и т. д.).
a) Насколько большим должен стать объем таких данных, чтобы вы наконец решили сохранить их в базе данных?
b) Какие изменения в вашем подходе к использованию этих данных подтолкнули бы вас к перемещению их в систему баз данных?

1.4. Если вы знакомы с особенностями систем управления версиями (например, Git или Subversion), сравните их возможности с возможностями систем баз данных.
a) Есть ли в системе управления версиями понятие записи?
b) Как операции извлечения/отправки версий соответствуют управлению параллельным доступом в базе данных?
c) Как выполняется фиксация изменений? Как производится отмена незафиксированных изменений?
d) Многие системы управления версиями сохраняют изменения в виде *файлов различий*, имеющих небольшой размер и описывающих, как преобразовать предыдущую версию файла с кодом в новую. Когда пользователь запрашивает текущую версию файла, система извлекает исходную версию и применяет к ней все файлы различий. Насколько хорошо эта стратегия удовлетворяет потребностям систем баз данных?

Практика

1.5. Выясните, используется ли в вашем учебном заведении или организации система баз данных. Если да:
a) кто из сотрудников напрямую использует систему баз данных в своей работе? (К их числу не относятся сотрудники, запускающие программы, которые используют базу данных без их ведома.) Для чего они ее используют?
b) когда пользователю требуется сделать с данными что-то, чего прежде он не делал, кто пишет запрос? Он сам или кто-то?

1.6. Установите и запустите серверы Derby и SimpleDB.
a) Запустите программы `ij` и `SimpleIJ` на компьютере, играющем роль сервера.
b) Если у вас есть второй компьютер, попробуйте запустить на нем демонстрационные клиентские программы и подключиться к серверу с этого компьютера.