1973
Программист-навигатор

Чарльз В. Бахман

Ричард Каннинг, председатель комитета по присуждению Тьюринговской премии 1973 г., представляя эту лекцию 28 августа на ежегодной конференции Ассоциации вычислительных машин (ACM) в Атланте, сказал следующее.

В последние пять — восемь лет в сфере вычислительной техники произошли важные изменения в области обработки данных. В самом начале развития информатики данные были тесно связаны с использующими их прикладными программами. Теперь ясно, что нужно разорвать эту связь. Мы хотим иметь данные, не зависящие от того, какая прикладная программа их использует, — т. е. данные, организованные и структурованные таким образом, чтобы они могли использоваться во многих приложениях и многими пользователями. То, к чему мы стремимся, — это *база данных.*

Работа по созданию баз данных сейчас находится в периоде младенчества. Но тем не менее уже известно от 1000 до 2000 систем управления базами данных, работающих по всему миру. Очень вероятно, что через десять лет число таких систем будет исчисляться десятками тысяч. Именно из-за большого количества инсталлированных систем влияние баз данных обещает быть огромным.

Человек, получающий премию Тьюринга в этом году, является настоящим пионером в области технологии создания баз данных. Ни один человек не повлиял так, как он, на развитие этой части информатики. Я выбрал три важнейших примера того, что он сделал.

Он был создателем и главным архитектором первой поступившей на рынок системы управления базами данных — The Integrated Data Store, разработанной в период с 1961 по 1964 г. [1, 2, 3, 4]. В настоящее время I-D-S является одной из наиболее широко используемых систем управления базами данных. Он был также одним из членов — учредителей группы по созданию базы данных CODASYL и работал в этой группе с 1964 по 1968 г. Технические требования, разработанные этой группой, выполняются и сейчас многими разработчиками

программ в различных частях света [5, 6]. Фактически сейчас эти требования являются единственной общепринятой основой архитектуры систем управления базами данных. Именно благодаря этому человеку эти требования после продолжительных дебатов и дискуссий воплотили многие из первоначальных идей I-D-S. И в-третьих, этот человек является создателем мощного метода представления отношений между данными — инструмента как для разработчиков баз данных, так и для разработчиков прикладных программ [7, 8].

Таким образом, его исследования продемонстрировали союз воображения и практики. Его глубокие работы уже оказали и будут оказывать значительное влияние на развитие информатики.

Я очень рад вручить премию Тьюринга 1973 г. Чарльзу В. Бахману.

Коперник полностью изменил наши взгляды на астрономические явления, предположив, что Земля вращается вокруг Солнца. Растет уверенность в том, что люди, занимающиеся обработкой информации, сильно выиграли бы, если бы восприняли радикально новую точку зрения, которая освобождает мышление прикладного программиста от централизма запоминающего устройства на магнитных сердечниках и дает ему свободу действовать как навигатору в базе данных. Для этого ему нужно, во-первых, освоить разнообразные навигационные умения и, во-вторых, выучить «правила движения», чтобы избежать конфликта с другими программистами, когда они будут одновременно находиться в информационном пространстве базы данных.

Новая точка зрения внесет такую же смуту в ряды программистов, какую в свое время гелиоцентрическая теория внесла в ряды старинных астрономов и теологов.

В этом году весь мир празднует пятисотлетие со дня рождения Николая Коперника, знаменитого польского астронома и математика. В 1543 г. он опубликовал свою книгу «Об обращении небесных сфер», в которой изложил новую теорию относительного движения Земли, планет и Солнца. Она была в прямом противоречии с геоцентрической теорией Птолемея, появившейся на 1400 лет раньше.

Коперник создал гелиоцентрическую теорию, основная идея которой состоит в том, что планеты движутся вокруг Солнца по круговым орбитам. Эта теория в течение долгого времени подвергалась сильнейшей критике. Приблизительно через сто лет Галилею пришлось предстать перед судом инквизиции в Риме, и он был вынужден заявить, что больше не верит в теорию Коперника. Но даже это не успокоило инквизиторов, и Галилей был приговорен к бессрочному тюремному заключению, а книга Коперника была помещена в список запрещенных книг, в котором и находилась в течение последующих двухсот лет.

Я вспоминаю о Копернике сегодня, чтобы проиллюстриро-

вать параллель, которая, как я верю, существует в мире вычислительной техники, или, точнее, в мире информационных систем. Последние 50 лет мы работали с информационными системами птолемеевского типа. Эти системы и большинство размышлений о таких системах основывались на компьютероцентрических идеях. (Я не случайно говорю о 50, а не о 25 годах нашей истории, потому что я считаю, что современные информационные системы берут начало от первых счетно-аналитических машин (табуляторов), а не от первых вычислительных машин с запоминаемой программой.) Люди античности считали само собой разумеющимся, что Солнце вращается вокруг Земли; точно так же и люди античного периода развития информационных систем считали само собой разумеющимся, что через счетно-аналитическую машину должен проходить последовательный файл. Каждая из этих точек зрения являлась адекватной моделью действительности для того времени и места, когда и где была создана. Но через некоторое время каждая из них оказалась некорректной и неадекватной, и ее пришлось заменить на другую модель, которая более точно описывает окружающий мир и его поведение.

Коперник познакомил мир с новой точкой зрения и заложил фундамент современной небесной механики. Этот взгляд создал основу для понимания путей Солнца и планет по небу, ранее казавшихся таинственными. Новая основа для понимания существует и в области создания информационных систем. Она состоит в повороте от компьютероцентрических представлений к представлениям, в центре которых находится база данных. Новое понимание приведет к новому решению наших проблем, связанных с построением баз данных, и ускорит покорение n-мерных структур данных, являющихся наилучшей моделью сложностей реального мира.

Самые первые базы данных, реализованные с помощью технологии последовательного доступа к файлам, в которых носителями информации были еще перфокарты, не сильно изменились, когда их перенесли с перфокарт сначала на магнитную ленту, а затем на магнитный диск. Единственное, на что это хоть как-то повлияло, были размер файлов и скорость работы с ними.

Для технологии последовательного доступа к файлам техника поиска хорошо развита. Нужно начать со значения первичного ключа интересующей нас записи и считывать в основной памяти все записи подряд до тех пор, пока не встретится нужная запись или запись с большим значением первичного ключа. (Первичный ключ — это область в записи, которая делает эту запись уникальной внутри данного файла.) Первичными ключами являются номера ценных бумаг, налоговых кви-

танций, страховых свидетельств, банковских счетов. Почти все они без исключения являются искусственными атрибутами, специально предназначенными для удостоверения уникальности. Естественные атрибуты, как, например, имена людей, географические названия, даты, времена и количества, не являются достоверно уникальными и поэтому не могут быть использованы с этой целью.

Появление запоминающих устройств прямого доступа стало причиной переворота в мышлении, похожего на коперниковский. Направления, обозначаемые словами «в» и «из», поменялись на обратные. Если директива «ввод» в мире последовательного доступа к файлам означала «с ленты в вычислительную машину», то новая директива «ввод» означает «в базу данных». Эта революция в мышлении превращает программиста из неподвижного наблюдателя объектов, проходящих перед ним в памяти машины, в активного навигатора, который может перемещаться по базе данных вдоль и поперек по своему желанию.

Кроме того, с появлением запоминающих устройств с прямым доступом появились новые способы поиска записей по первичному ключу. Первый из них был назван рандомизацией или хэшингом. В нем происходит обработка первичного ключа с помощью специального алгоритма, который определяет область памяти, в которой с наибольшей вероятностью может находиться искомая запись. Если запись в этой области не найдена, применяется алгоритм переполнения (overflow algorithm) для нахождения других областей, в которых может храниться эта запись, если она вообще существует. Переполнение возникает при занесении записи в базу данных, если в этот момент область, где должна была бы находиться запись, оказывается заполненной.

Как альтернатива методу рандомизации был создан метод индексно-последовательного доступа. Для управления хранением и поиском записей в нем также применяются первичные ключи, но путем использования многоуровневых индексов.

Программист, перейдя от последовательной обработки файлов к хэшингу или к индексно-последовательному доступу, значительно уменьшает время доступа, ибо он теперь может добраться до интересующей его записи, не проходя последовательно всех предыдущих записей в файле. Однако он до сих пор находится в одномерном мире, так как имеет дело с одним первичным ключом, являющимся единственным инструментом для управления доступом.

С этого места я хочу начать обучение «полностью оперившегося» программиста-навигатора ориентированию в n-мерном пространстве данных. Но прежде, чем я смогу понятно опи-

сать этот процесс, я хочу пояснить, что же означают слова «управление базой данных».

Это понятие включает в себя все аспекты хранения, поиска, изменения и удаления данных в файлах, касающихся персонала или продукции, покупки авиабилетов или лабораторных экспериментов, — данных, которые используются неоднократно и обновляются всякий раз, как только становится доступной новая информация. Эти файлы с помощью некоторой запоминающей структуры и соответствующих драйверов преобразуются в информацию, носителями которой является магнитная лента или пакет дисков.

Управление базой данных включает в себя два основных вида деятельности. Первый — обработка запросов и поиск информации — обеспечивает доступ к сохраненным ранее данным, чтобы определить запомненное состояние некоей сущности или отношения в реальном мире. Эти данные могли быть записаны другой задачей несколько секунд, минут, часов или даже дней назад и были сохранены системой управления базой данных. Постоянной обязанностью системы управления базой данных является хранение информации с того момента, когда она была записана, до тех пор, когда она будет востребована запросом. Обработка запросов служит для того, чтобы находить информацию, необходимую для принятия решений.

Частью деятельности по обработке запросов является формирование сообщений. В давние годы запоминающих устройств с последовательным доступом и пакетной обработкой данных не было другой альтернативы, кроме создания и вывода на печать файлов в формате сообщений. Спонтанные требования просмотреть банковский счет, инвентарную опись или план выпуска продукции не могли быть выполнены эффективно, потому что для отыскания каких-либо данных приходилось просматривать весь файл. Удельный вес такой формы обработки запросов уменьшается, и со временем она исчезнет совсем, кроме как для хранения архивов или для удовлетворения аппетитов паркинсоновской бюрократии.

Вторым видом деятельности при управлении базой данных является обновление данных, что включает в себя ввод информации в базу данных, ее многократные изменения и, наконец, удаление ее из системы, когда она перестанет быть нужной.

Обновление данных является реакцией на те изменения в реальном мире, которые должны быть зарегистрированы. Прием на работу нового сотрудника приведет к появлению новой записи, которая должна быть сохранена. Уменьшение количества товаров на складах приведет к изменению инвентарной описи. Отмена предварительного заказа на билет на самолет

приведет к удалению записи. Все эти данные должны храниться и обновляться в ожидании следующих запросов.

Для сортировки файлов требовалось большое количество компьютерного времени. Сортировка файлов использовалась при сортировке трансакций, предшествующей пакетному последовательному обновлению базы данных, и при подготовке сообщений. Переход к обновлению базы данных в режиме оперативной обработки трансакций и к немедленному исполнению запросов на извлечение информации из базы данных и на подготовку отчетов уменьшает необходимость сортировки на уровне файлов.

А теперь давайте вернемся к нашему рассказу о программисте-навигаторе. Мы покинули его, когда он пытался воспользоваться методами рандомизации или индексно-последовательного доступа, основанными на использовании первичных ключей, чтобы ускорить процесс поиска и обновления информации в файле.

В дополнение к первичным ключам записей полезно иметь возможность находить запись по значениям некоторых других полей. Например, при планировании премий за 10 лет работы может понадобиться найти все такие записи о служащих, у которых значение поля «год приема на работу» было бы равно 1964. Такой метод доступа называется поиском по вторичному ключу. Количество записей, которые будут найдены по значению вторичного ключа, непредсказуемо и может изменяться от нуля до общего количества записей в файле. В противоположность этому по первичному ключу может быть найдена максимум одна запись.

С появлением поиска по вторичным ключам пространство данных, бывшее одномерным, приобретает дополнительные измерения, число которых равно числу полей в записи. Для файлов небольшого или среднего размера система управления базой данных может снабдить указателем каждое поле записи. Такие полностью индексированные файлы называются инвертированными файлами. Однако в больших активных файлах неэкономно индексировать каждое поле. Поэтому разумно выбрать те поля, содержимое которых будет наиболее часто использоваться, в качестве ключей (признаков, критериев) для поиска, и создать вторичные индексы только для этих полей.

Различия между файлом и базой данных не слишком ясны. Однако одно из них имеет непосредственное отношение к теме нашего разговора. В базах данных обычно бывает несколько или много различных типов записей. Например, в базе данных, содержащей сведения о персонале, могут быть записи с информацией о сотрудниках, различных подразделениях, квалификации, удержаниях из жалованья, послужном списке и

образовании. Каждый тип записи имеет свой собственный уникальный первичный ключ, а все остальные ее поля являются потенциальными вторичными ключами.

В такой базе данных первичные и вторичные ключи находятся в интересных взаимоотношениях, когда первичный ключ записи одного типа играет роль вторичного ключа для записи другого типа. Возвращаясь к нашему примеру базы данных со сведениями о персонале, заметим, что поле с названием «код подразделения» встречается как в записях, относящихся к сотрудникам, так и в записях, относящихся к подразделениям. Это поле является одним из возможных вторичных ключей записей, относящихся к сотрудникам, и единственным первичным ключом записей о подразделениях.

Равноправность полей, содержащих первичные и вторичные ключи, отражает сложные взаимосвязи в реальном мире и дает возможность воссоздать эти взаимосвязи при компьютерной обработке информации. Использование одного и того же значения в качестве первичного ключа для одной записи и вторичного ключа для некоторого множества других записей является основной концепцией для возникновения и развития структурных наборов данных. Создатели системы Integrated Data Store (I-D-S) и других систем, построенных на основе тех же принципов, считают, что основным преимуществом этих систем для программиста является возможность объединять записи в структурные наборы данных, которые можно потом использовать как пути поиска. Все реализованные системы, созданные под руководством COBOL Database Task Group, являются системами этого класса.

Переход от системы файлов, каждый из которых содержит один тип записи, к базе данных с разными типами записей и структурированными наборами данных дает множество преимуществ. Одним из них является значительное увеличение эффективности, обусловленное использованием для поиска всех записей с определенным значением ключа структурированных множеств данных одновременно с первичными и вторичными ключами. При использовании структурированных наборов данных можно исключить всю избыточную информацию, что приводит к уменьшению необходимого объема памяти. Если избыточные данные были специально сохранены, чтобы увеличить скорость поиска, то их можно использовать, чтобы убедиться, что обновление какого-либо значения в одной записи влечет за собой изменения во всех соответствующих записях. Возможность так называемой «кластеризации» (объединения в кластеры), когда запись-хозяин и некоторые или большая часть записей данного множества физически хранятся в одном блоке или на одной странице, также увеличивает эффек-

тивность доступа. Системы такого типа работают с использованием виртуальной памяти с 1962 г.

Другое важное качественное преимущество базы данных — возможность выбирать порядок поиска записей внутри множества: по объявленному полю сортировки или же по времени ввода.

Чтобы подчеркнуть роль программиста как навигатора, давайте перечислим возможности поиска записи, которые имеются в его распоряжении. Это команды, с которыми он может обратиться к базе данных — один раз, многократно или комбинируя одну с другой, — когда он пробирается среди данных, чтобы ответить на запрос или обновить информацию.

1. Он может начать с начала базы данных или с любой известной ему записи и последовательно просматривать «следующие» записи до тех пор, пока не доберется до интересующей его записи или до конца.

2. Он может войти в базу данных с помощью ключа, обеспечивающего прямой доступ к физическому местоположению записи. (Ключ базы данных — это постоянный адрес в виртуальной памяти, приписанный записи при ее создании.)

3. Он может войти в базу данных по значению первичного ключа. (Как хэшинг, так и метод индексно-последовательного доступа дадут одинаковый результат.)

4. Он может войти в базу данных по значению вторичного ключа и последовательно просмотреть все записи, для которых в соответствующем поле стоит определенное значение вторичного ключа.

5. Он может начать с записи-хозяина некоторого множества и последовательно обратиться ко всем элементам этого множества. (Это эквивалентно превращению первичного ключа базы данных во вторичный.)

6. Он может начать с любого элемента множества и получить доступ как к следующему, так и к предыдущему элементу в этом множестве.

7. Он может начать с любого элемента множества и обратиться к записи-хозяину этого множества, преобразуя таким образом вторичный ключ базы данных в первичный.

Каждый из этих способов доступа интересен сам по себе, и все они очень полезны. Однако именно согласованное применение всех методов дает программисту широкие возможности войти в большую базу данных и перемещаться в ней, обращаясь только к тем записям, которые ему нужны при обработке запросов и обновлении информации для последующих запросов.

Чтобы проиллюстрировать, как обработка единственной трансакции может включать в себя определение пути внутри

базы данных, вообразив себе следующий сценарий. Трансакция содержит первичный ключ или просто ключ записи в базе данных, нужный для того, чтобы найти точку всхода в базу данных. Эта запись будет использована, чтобы обратиться к другим записям (как к записи-хозяину, так и к элементам) множества. Каждая из этих записей в свою очередь может быть использована как точка выхода, чтобы получить доступ к другому множеству.

Например, рассмотрим требование составить список сотрудников какого-либо подразделения, если известен его код. Такой запрос может быть обращен к базе данных, содержащей только два типа записей: записи, содержащие сведения о сотрудниках, и записи, содержащие сведения о подразделениях. Для простоты будем рассматривать записи о подразделениях, в которых только два поля: код отдела, являющийся первичным ключом, и название отдела, которое является описанием. Рассмотрим также персональные записи, состоящие из трех полей, а именно: номер сотрудника по списку является первичным ключом записи, фамилия и имя сотрудника являются описанием, а код отдела, где работает сотрудник, является вторичным ключом, который управляет выбором множества и расположением записей в множестве. Одновременное использование кода отдела в обоих типах записей и объявление множества по этому ключу являются основой для создания и поддержания множественных отношений между записью, содержащей информацию о подразделении, и всеми персональными записями, содержащими сведения о сотрудниках этого подразделения. Таким образом, использование множества персональных записей обеспечивает механизм для того, чтобы после отыскания по первичному ключу соответствующей ему записи с информацией о конкретном подразделении немедленно выдать список сотрудников этого подразделения. Для этого не требуется обращаться ни к какой другой записи.

Добавление к записи о подразделении номера сотрудника, который руководит этим подразделением, в значительной мере расширяет навигационные возможности и служит основанием для создания множеств еще одного класса. При обращении к множеству из этого класса вызываются все записи о подразделениях, которыми руководит определенный сотрудник. Теперь единственное число — номер сотрудника или код отдела — дает точки входа в интегрированную структуру данных о предприятии. Если известны номер сотрудника и множество, состоящее из записей о подразделениях, которыми он руководит, то можно выдать список этих подразделений. Далее можно выдать список сотрудников каждого такого подразделения. Запрос о подразделениях, руководимых каждым из этих сотруд-

ников, можно повторять до тех пор, пока не будут перечислены все подчиненные сотрудники и подразделения. С другой стороны, с помощью той же самой структуры данных можно определить руководителя руководителя, руководителя руководителя руководителя и так далее — до президента компании.

Программиста, который научился действовать в n-мерном пространстве данных, ожидает еще немало опасностей и приключений. Как навигатор, он должен преодолевать плохо различимые мели и рифы в море информации, которые возникают потому, что он должен ориентироваться в общем пространстве базы данных.

Совместный доступ к базе данных — это новая и сложная разновидность мультипрограммного режима работы или режима работы с разделением времени, которые были созданы, чтобы обеспечить совместное, но независимое использование ресурсов вычислительной машины. При работе в мультипрограммном режиме программист, решающий свою задачу, не знает и не заботится о том, что его программа может быть не единственной в памяти компьютера, так как он уверен, что его адресное пространство не зависит от адресного пространства других программ. Обеспечить сохранность каждой программы и наилучшим образом использовать память, процессор и другие ресурсы — задача операционной системы. Совместный доступ — это специализированный вариант мультипрограммного режима работы, при котором совместно используемыми ресурсами являются записи в базе данных. Записи базы данных коренным образом отличаются от таких ресурсов, как оперативное запоминающее устройство или процессор, потому что их поля изменяют свои значения в процессе обновления данных и не возвращаются потом в исходное состояние. Поэтому задача, неоднократно использующая определенную запись базы данных, может обнаружить, что ее содержание или отношение принадлежности к множеству изменилось со времени предыдущего обращения. В результате алгоритм, осуществляющий сложные вычисления, может обращаться к довольно-таки нестабильной информации. Представьте себе, что вы решаете задачу с помощью сходящегося итерационного процесса, а переменные меняются случайным образом! Вообразите, что вы хотите подсчитать промежуточный баланс, а тем временем кто-то совершает разнообразные операции со счетами! Представьте себе две конкурирующие задачи в системе заказа авиабилетов, которые одновременно пытаются продать последний билет на рейс!

Первая реакция здравомыслящего человека — мысль, что распределенный доступ — это нонсенс и о нем надо забыть. Однако серьезные причины вынуждают его использовать и

развивать. Быстродействие процессоров, которые доступны сегодня и будут доступны в обозримом будущем, значительно превосходит быстродействие используемых запоминающих устройств прямого доступа. Более того, даже если быстродействие (эффективность) запоминающих устройств станет сравнимым с быстродействием процессоров, останется еще две важные причины стремления к разработке систем совместного доступа. Первая — это тенденция к объединению большого количества целевых файлов в несколько интегрированных баз данных; вторая — это тенденция к интегрированной обработке данных, при которой процессор может выполнять работу только в том темпе, в каком ему позволяют вручную вводимые сообщения. При отсутствии совместного доступа вся база данных будет заблокирована до тех пор, пока не закончится выполнение командного (пакетного, бат-) файла или трансакции и их взаимодействие с пользователем.

Реализация современных запоминающих устройств сильно зависит от принятой схемы их использования. Если она представляет собой чередующуюся последовательность вида: обращение к памяти, обработка, обращение к памяти, обработка и т. д. — и выполнение каждого обращения зависит от интерпретации предыдущего, то эффективность будет очень мала. В мультипрограммном режиме генерируется много независимых обращений к памяти, и они могут быть исполнены параллельно, потому что направлены разным запоминающим устройствам. Более того, когда существует очередь обращений к одному и тому же запоминающему устройству, его производительность в настоящее время может быть значительно увеличена с помощью методов предварительной обработки данных, уменьшающих время поиска и ожидания. Возможность увеличения пропускной способности является очень весомым доводом в пользу развития систем совместного доступа.

Из двух основных функций системы управления базой данных — обработка запросов и обновления данных — только вторая может быть причиной затруднений при работе в режиме совместного доступа. Неограниченное количество задач может одновременно извлекать информацию из базы данных. Но как только одна из задач начинает изменять данные, возникает возможность ошибок. В результате обработки трансакции может потребоваться изменение всего нескольких записей из тысяч или миллионов, находящихся в базе данных. Поэтому сотни задач могли бы выполнять трансакции одновременно без возникновения конфликтных ситуаций. Однако наступит момент, когда две задачи одновременно захотят работать с одной записью.

Двумя основными причинами трудностей при реализации

совместного доступа являются интерференция и порча данных. Интерференция — это отрицательное влияние изменения информации, производимого одной задачей, на результат работы другой. Я уже приводил пример, иллюстрирующий проблему интерференции, когда одна задача подсчитывает промежуточный баланс, в то время как другая выдает трансакции, приводящие к изменению счетов. Если в процессе решения задачи на нее влияет другая, то для получения правильного результата первая задача должна быть прервана и запущена снова. Все выходные данные, полученные при предыдущем проходе, нужно удалить, так как будут выданы новые. *Порча данных* — это отрицательный эффект, который происходит в результате комбинации двух событий, а именно когда вторая задача прервана, а ее выходные данные (т. е. изменения в базе данных или какие-либо сообщения) уже считаны нашей первой задачей. Прерванная задача и ее выходная информация должны быть удалены из системы. Более того, задачи, «испорченные» этой информацией (на работу которых повлияла эта информация), должны быть также прерваны и запущены повторно, чтобы данные, к которым они обращаются, были корректны.

Критическим вопросом при решении проблем совместного доступа является «глубина видимости», которую следует предоставить прикладному программисту. Версия I-D-S с совместным доступом, разработанная Weyerhaeuser Company, основывается на предпосылке, что программист вообще не должен знать о проблемах совместного доступа. В этой системе автоматически блокируются каждая измененная запись и каждое сообщение, сформированное задачей, до тех пор, пока не произойдет нормальное окончание этой задачи. Таким образом проблема порчи данных полностью снимается. Побочным эффектом такого динамического блокирования информации является возможность возникновения взаимоблокировки (зависания), когда две или больше задач ждут окончания работы друг друга, чтобы получить доступ к желаемой записи. Система I-D-S реагирует на возникновение взаимной блокировки следующим образом: она прерывает задачу, послужившую причиной этой ситуации, восстанавливает записи, измененные этой задачей, и делает эти записи доступными для всех задач, находящихся в состоянии ожидания. Затем перезапускается прерванная задача.

Возникают ли в действительности ситуации взаимоблокировки? По последним сведениям в системе Weyerhaeuser, ориентированной на обработку трансакций, около 10% всех запущенных задач бывают прерваны из-за взаимоблокировки. За час выбрасываются и перезапускаются приблизительно 100 задач. Ужасно ли это? Слишком ли неэффективно? На эти во-

просы трудно ответить, так как наши представления о стандартах эффективности работы таких систем четко не определены. Более того, результаты зависят от применений. Эффективность системы Weyerhaeuser I-D-S составляет 90% в терминах успешно завершенных заданий. Однако действительно важными вопросами являются следующие:

— К увеличению или к уменьшению количества задач, успешно завершенных в течение часа, привел бы отказ от совместного доступа?

— Не была ли бы более эффективной другая стратегия, основанная не на том, чтобы избегать порчи данных, а на том, чтобы своевременно определять такие ситуации?

— Не повысится ли эффективность, если программист будет знать о проблемах совместного доступа и учитывать это обстоятельство в своих программах?

Все эти вопросы становятся насущными для программиста-навигатора и для людей, которые создают и развивают средства навигации.

Я считаю, что сейчас для прикладного программиста настало время отказаться от представлений, в которых центром всего является память компьютера, принять вызов и воспользоваться возможностями навигации в n-мерном пространстве информации. Системы математического обеспечения для поддержки таких возможностей существуют сегодня и становятся все более доступными.

Бертран Рассел, знаменитый английский математик и философ, однажды сказал, что теория относительности потребовала изменений в нашей воображаемой картине мира. Похожие изменения должны произойти и в нашей воображаемой картине мира информационных систем.

Основная проблема — это переориентация мышления людей, занимающихся проблемами обработки информации. Это не только программисты, но создатели прикладных систем, ставящие задачи для прикладного программирования, и системные программисты, которые будут придумывать и реализовывать операционные системы, системы обмена сообщениями и системы управления базами данных завтрашнего дня.

Коперник заложил основы небесной механики более 400 лет назад. Это та самая наука, которая дает минимальные с точки зрения энергии решения при планировании траектории полетов к Луне и другим планетам. Нужно развивать такой же научный подход, результатом которого были бы энергетически минимальные решения задачи доступа в базах данных. Эта проблема вдвойне интересна, так как она включает в себя как задачи перемещения по уже существующим базам данных, так и вопрос, как построить базу данных и затем ее изменять, что-

бы она наилучшим образом соответствовала изменяющимся моделям доступа. Вы можете себе представить реконструкцию Солнечной системы с целью минимизировать время полета между планетами?

Необходимо, чтобы изучение механизмов структурирования информации развивалось как инженерная дисциплина, основанная на главных принципах проектирования. Необходимо, чтобы оно могло стать учебной дисциплиной и ее преподавали. Стоимость оборудования для баз данных, которые будут инсталлированы к 1980 г., оценивается в 100 млрд. долл. (на основе данных за 1970 г.). Далее, ожидается, что отсутствие эффективной стандартизации добавит еще 20% или 20 млрд. долл. к этой сумме. Следовательно, было бы разумнее отказаться от консерватизма, эмоций и теологических аргументов, замедляющих прогресс в настоящее время. В университетах проблемы структурирования информации в значительной степени игнорировались и предпочтение отдавалось задачам, которые лучше подходят для курсовых и дипломных работ. Создание большой базы данных — это дорогостоящая программа, и университетский бюджет просто не может себе позволить ее финансирование. Поэтому для финансирования и обеспечения необходимых для достижения прогресса мощностей нужно создавать совместные программы для университетов и промышленности или для университетов и правительственных учреждений. В системе Weyerhaeuser содержится достаточно материала для полудюжины диссертаций, который ждет, чтобы кто-нибудь пришел и откопал его. Я не имею в виду исследования по созданию новых алгоритмов рандомизации. Я имею в виду исследования по структурированию около миллиарда знаков информации о реальном бизнесе, организованных в самую безупречную из структур данных, известных на сегодняшний день.

Проблемой является и публикация технической литературы. Проще всего опубликовать свою работу в периодических выпусках тематических групп SIGBDP и SIGFIDET Ассоциации вычислительных машин (ACM). Правила реферирования статей и обычная практика публикации в журнале Communications of the ACM приводят к тому, что от получения статьи до ее издания проходит от одного года до полутора лет. Прибавьте к этому время, необходимое автору для подготовки своих идей к печати, и вы получите задержку между получением важных результатов и первой их публикацией по меньшей мере в два года.

Может быть, самым большим препятствием для прогресса является отсутствие у большей части пользователей доступной информации о базах данных, которое является следствием то-

го, что интересы рынка доминируют над интересами отдельного пользователя. Если бы программисты предоставили свой опыт, требования и средства решения задач для действительно открытого обмена информацией, изменения происходили бы гораздо быстрее. Последяя акция SHARE — открыть членство для всех организаций, продающих программы, и всех пользователей — является важным шагом вперед. Рабочая конференция по системам управления базами данных, спонсором которой был SHARE, стала ареной свободной дискуссии, в которой пользователи всех видов оборудования и всевозможных баз данных могли рассказать о своем опыте и своих нуждах.

Все расширяющийся диалог начался. Я очень хочу, чтобы он продолжался, и надеюсь, что так и будет. Я уверен, что если продолжать в том же духе, если никто не будет стремиться доминировать в области идей, то мы сможем обеспечить программиста эффективными средствами навигации.

ЛИТЕРАТУРА

1. A general purpose programming system for random access memories (with S. B. Williams). Proc. AFIPS 1964 FJCC, Vol. 26, AFIPS Press, Montvale, N. J., pp. 411—412.
2. Integrated Data Store. DPMA Quarterly (Jan. 1965).
3. Software for random access processing. Datamation (Apr. 1965), 36—41.
4. Integrated Data Store — Case Study. Proc. Sec. Symp. on Computer-Centered Data Base Systems sponsored by ARPA, SDC, and ESD, 1966.
5. Implementation techniques for data structure sets. Proc. of SHARE Working Conf. on Data Base Systems, Montreal, Canada, July 1973.
6. The evolution of Data Structures. Proc. NordDATA Conf., Aug. 1973, Copenhagen, Denmark, pp. 1075—1093.
7. Data structure diagrams. Data Base 1, 2 (1969), Quarterly Newsletter of ACM SIGBDP, pp. 4—10.
8. Set concepts for data structures. In Encyclopedia of Computer Science, Amerback Corp. (to be published in 1974).

ПОСТСКРИПТУМ

ПРОГРАММИСТ КАК НАВИГАТОР, АРХИТЕКТОР, СВЯЗИСТ, СОЗДАТЕЛЬ МОДЕЛЕЙ, СОТРУДНИК ЭКСПЕРТНОЙ СИСТЕМЫ И РУКОВОДИТЕЛЬ

Чарльз В. Бахман
Bachman Information Systems, Inc.

С момента написания Тьюринговской речи, которая называлась «Программист-навигатор», прошло 13 лет. Использование баз данных стало обычным, даже популярным занятием. Некоторые программисты используют средства навигации. Остальные пытаются это делать. Я приложил значительные усилия, чтобы доказать преимущества сетевой модели данных и ее расширения с целью увеличить ее выразительные возможности [1, 2, 3, 4]. Горячие и яростные споры и дискуссии, касающиеся моделей данных, к настоящему времени в значительной степени остались позади. Сейчас единст-

венное, с чем все согласны — тот факт, что можно с пользой работать с базами данных, основанными на любой из общепринятых моделей данных, и даже с теми, которые не имеют явного отношения ни к какой конкретной модели данных.

ПРОГРАММИСТ КАК АРХИТЕКТОР

Изучение архитектуры компьютерных информационных систем далеко продвинулось за это время. Два проекта, каждый из которых по-своему важен, способствовали привлечению внимания к этому вопросу. Группа ANSI/X3/SPARC по изучению систем управления базами данных (1972—1977 гг.) опубликовала свои исследования [5] по архитектуре систем хранения и поиска данных. Это была одна из первых попыток четко понять и описать различные уровни работы человека и программного обеспечения в процессе хранения и поиска информации. Более того, были выделены и описаны интерфейсы между различными модулями математического обеспечения и людьми, которые с ними работают (администраторами, создателями баз данных и программистами). Очень важно, что в этой работе описаны как административный интерфейс, так и интерфейс времени исполнения. Этот проект способствовал развитию понятия концептуальной схемы как описания информационных структур, сделанного на более высоком уровне абстракции и не зависящего от способа представления информации.

ПРОГРАММИСТ КАК СВЯЗИСТ

Подкомитет ISO/TC97/SC16 Международной организации по стандартизации (ISO) создал (1979—1982 гг.) Эталонную модель взаимосвязи открытых систем. Эталонная модель — это эталонная схема архитектуры систем обмена информацией, разработанная в виде международного стандарта [7], чтобы она служила контролирующей и объединяющей моделью для последующей серии более подробных стандартов. В этой архитектуре определяется семь иерархических уровней обработки данных, поддерживающих обмен между прикладными процессами. Каждый уровень описывается в терминах «административных логических объектов»[1], «обрабатывающих логических объектов», «услуг» и «протоколов».

Для обрабатывающих логических объектов каждого уровня должны быть выделены и стандартизованы следующие четыре вида интерфейсов:

(1) услуги, которые обрабатывающий объект предоставляет обрабатывающим объектам, лежащим на один уровень выше;
(2) протокол обмена, с помощью которого обрабатывающий объект общается с другими обрабатывающими объектами одного с ним уровня;
(3) использование обрабатывающими объектами данного уровня услуг, предоставляемых обрабатывающими объектами, лежащими на один уровень ниже;
(4) административный протокол, посредством которого данный обрабатывающий объект контролируется административными объектами своего уровня.

---

[1] Сочетание «логический объект» употребляется в стандарте ISO/TC97 для обозначения активного элемента, который играет некую роль в процессе обмена. Я использовал прилагательные «обрабатывающий» и «административный» для различия объектов, действующих во время обмена и при инициализации (set-up-time entities). Такое использование слова «объект» отличается от принятого при моделировании данных, где «объект» означает что-то, что существует и о чем что-либо известно.

Подробные стандарты, разработанные последовательно для каждого уровня, касаются индивидуальных протоколов, услуг и использования услуг.

Глубину предвидения и масштаб этой работы можно в какой-то мере оценить, вспомнив некоторые из дискуссий об адресуемости (addressability). Как велико должно быть адресное пространство, чтобы определить все обрабатывающие логические объекты, которые могут захотеть обмениваться информацией? В результате одной из дискуссий возник следующий сценарий:

К концу 2000 года численность населения Земли составит около 10 миллиардов человек (10 миллиардов адресов).

Предположим, что каждого из этих людей будет обслуживать в среднем 100 роботов (1 триллион адресов).

Для учета непредвиденных случайностей, а также продолжительности срока полезного использования адресного пространства в 25 лет, умножим еще на 10 (10 триллионов адресов).

Будем предполагать, что присвоение адресов происходило в ходе политических процессов начиная с образования Организации Объединенных Наций, и что 99 процентов адресов на самом деле недоступно для обмена на уровне прикладных программ (1 квадрильон адресов).

Следовательно, один квадрильон адресов — это правильный порядок величины адресного пространства для рассмотренного случая. В десятичной системе такое число представляется единицей с 15 нулями, в двоичной это единица и приблизительно 50 нулей.

В этом году работа ISO по созданию стандартов для взаимосвязи открытых систем получила дополнительную поддержку в результате создания в Соединенных Штатах Корпорации по открытым системам (Corporation for Open Systems). COS — это организация, деятельность которой охватывает всю индустрию создания и использования баз данных: пользователей, связистов и производителей. Она образована для поддержки выполнения стандартов ISO и для того, чтобы новое или модифицированное оборудование могло быть проверено на соответствие стандартам ISO.

Автор, делая доклад для технического комитета ISO/TC97 в качестве председателя подкомитета ISO/TC97/SC16, рекомендовал комитету TC97 разрабатывать «эталонную модель для автоматизированных информационных систем» [8, 9]. Эту расширенную модель можно было бы использовать, чтобы включить всю работу комитета ISO/TC97 по автоматическим и информационным системам в перспективу дальнейшего развития и таким образом выделить наиболее критические с точки зрения стандартизации ее участки.

В 1984—1985 гг. структура подкомитетов ISO/TC97 изменилась, и был создан новый подкомитет ISO/TC97/SC21, который взял на себя не только все функции SC16, но и дополнительные обязанности, касающиеся архитектуры систем хранения и поиска информации. Со временем в его компетенцию будут входить также вопросы сохранности данных и их защищенности от несанкционированного доступа, потому что невозможно создать законченную схему архитектуры систем хранения, поиска и обмена информацией в отрыве от проблем сохранности данных и их защиты (секретности).

ПРОГРАММИСТ КАК СОЗДАТЕЛЬ МОДЕЛЕЙ

За эти 13 лет я потратил довольно много времени на расширение работ над концептуальной схемой в группе по изучению СУБД (ANSI/SPARC Study Group on DMBS), сочетая эту деятельность с работой в области обмена информацией и методов формальных описаний. Вначале работа над

концептуальной схемой ограничивалась только информацией и форматами данных, касающимися бизнеса, как теми, которые хранятся в файлах и базах данных (внутренняя схема), так и представленными в программах (внешняя схема). Моей целью было расширить эту абстракцию и включить в нее описания всех активных агентов (людей, компьютерных программ и физических процессов), использующих информацию, а также каналов связи, которыми они пользуются, и сообщений, которыми они обмениваются.

Я стремился еще больше расширить эту абстракцию и включить в нее правила, которым подчиняется поведение объектов (агентов), использующих информацию. Такие расширенные концептуальные схемы получили название моделей предприятия (enterprise models) или моделей бизнеса (business models).

Зачем строить модель бизнеса? Во-первых, она является таким способом описания требований к организации процесса обработки информации, который одинаково понятен и пользователям, и людям, занимающимся обработкой информации. Во-вторых, она нужна как основа для автоматизации процесса создания прикладного программного обеспечения. Прикладным программным обеспечением я называю совокупность, включающую в себя описания базы данных и файлов, прикладные программы, а также контрольные параметры оборудования, необходимые для инсталляции необходимых файлов и программ и управления их работой.

Операция перевода модели бизнеса в множество прикладных программ, реализующих и поддерживающих эту модель — это операция, переводящая вопрос «что?» в мире бизнеса в вопрос «как?» в мире компьютеров и обмена информацией. Для такого перевода требуется три дополнительных элемента, лежащих за пределами формального определения модели бизнеса:

1. Нужна информация о количествах, скоростях и временах отклика как параметрах, которым должна удовлетворять модель.

2. Нужна информация об имеющихся в наличии процессорах, запоминающих устройствах, средствах связи и информация об имеющихся компиляторах, СУБД, системах передачи информации, мониторах, обрабатывающих трансакции, и операционных системах.

3. Необходимы также экспертные знания для того, чтобы оценить рабочие параметры и эксплуатационные характеристики имеющегося в наличии оборудования и программного обеспечения, и затем определить способ его наиболее эффективного использования с учетом функциональных и количественных ограничений.

Такими экспертными знаниями по исполнению и оптимизации обладают конкретные люди — создатели баз данных, прикладные программисты, системные программисты. Лучшие из них превосходно справляются со своей работой, но результаты работы многих других разочаровывают. Все это дорого и занимает гораздо больше времени, чем хотелось бы.

ПРОГРАММИСТ КАК СОТРУДНИК ЭКСПЕРТНОЙ СИСТЕМЫ

Недостаток специалистов достаточно высокой квалификации побудил нас начать исследования по автоматизации работы создателей баз данных, прикладных и системных программистов. Такая автоматизация очень сложна, так как процесс перевода модели бизнеса в эффективно работающие прикладные программы не является вполне детерминированным. Часто существует несколько альтернативных подходов, каждый со своей динамикой развития и затратами. В этом случае необходимо проводить экспертизу и принимать решение. Эта трудность привела к тому, что при оценке инструментальных средств и оборудования используются методы, разработанные в мире искусственного интеллекта, в котором большое внимание уделяется областям, связанным с применением неполных знаний (imperfect knowledge).

В мире искусственного интеллекта с его программными комплексами,

основанными на использовании знаний, существует значительный опыт разработки интерактивных систем, в которых постоянно работающий человек-эксперт может сотрудничать с «клонированным» экспертом, содержащимся в комплексе программ, и решать задачи, которые трудно решить другим способом. Вместе они могут сделать все необходимое для перехода с абстрактного концептуального уровня на физический, учитывая все проблемы и возможности конкретной реализации.

ПРОГРАММИСТ КАК РУКОВОДИТЕЛЬ

Есть причины думать, что «клонированные» эксперты, содержащиеся в системах, использующих знания (экспертных системах), будут совершенствоваться со временем. В процессе этого совершенствования роль резидентного эксперта-человека будет изменяться, и из соавтора экспертной системы он будет превращаться в ее руководителя. Как руководитель он будет контролировать работу экспертной системы и проверять, все ли рабочие режимы и условия эксплуатации предусмотрены. После проверки и просмотра всех возможных модификаций эксперт как руководитель должен будет подписать окончательный проект, так же как руководитель группы инженеров подписывает результат работы своих подчиненных. При использовании информационных систем в области бизнеса ничего не может быть сделано без того, чтобы кто-нибудь это не проверил и не нес за это ответственность.

ЗАКЛЮЧЕНИЕ

Есть что-то поэтическое в том, что методы, использующиеся при создании баз данных, соединяются с методами из области искусственного интеллекта. Поэтическое, потому что уже первые упоминания (1960 г.) об обработке списков в литературе по системам искусственного интеллекта послужили основой для создания связных списков, которые были первым и до сих пор остаются самым часто используемым средством реализации баз данных. Путаница с понятием структурного набора данных и наиболее распространенным способом его воплощения внушает беспокойство. Существует много хорошо известных способов реализации структурных наборов данных [10], каждый со своими рабочими характеристиками, и все они обеспечивают адекватную поддержку функциональных качеств множества.

Интересно, удастся ли методам экспертной оценки реализаций из мира баз данных существенно повлиять на мир LISP'а и искусственного интеллекта, как это происходит в коммерческих применениях, где базы знаний велики и распределены по многочисленным взаимодействующим рабочим станциям с системами искусственного интеллекта. В этом случае эффективность и время отклика зависят от успешной работы совместно используемой системами с базами знаний распределенной виртуальной памяти.

ЛИТЕРАТУРА

1. Bachman, C. W. Why restrict the modeling capability of the CODASYL data structure sets? in Proceedings of the AFIPS National Computer Conference, vol. 46. AFIPS Press, Reston, Va., 1977.
2. Bachman, C. W., and Daya, M. The role concept in data models. In Proceedings of the 3rd Very Large Database Conference, 1977.
3. Bachman, C. W. The structuring capabilities of the molecular data model (partnership data model). In Entity-Relationship Approach to Software Engeneering. Elsevier Science, New York, 1983.
4. Bachman, C. W. The partnership data model. Presented at the Fall 1983 IEEE Computer Conference (Washington, D. C.).

5. ANSI/X3/SPARC/Study Group — Database Management Systems. Framework Report on Database Management Systems. AFIPS Press, Reston, Va., 1978.
6. ISO/TC97/SC5/WG3. Concepts and terminology for the conceptual schema. January 15, 1981.
7. ISO. Computers and Information Systems — Open Systems Interconnection Reference Model. Standard 7498. American National Standards Institute, New York, N. Y.
8. Bachman, C. W. The context of open systems interconnection within computer-based information systems. In Proceedings of Gesellschaft fur Informatik, Jan. 1980.
9. Bachman, C. W., and Ross, R. G. Toward a more complete reference model of computer-based information systems. J. Comput. Standards 1 (1982); also published in Comput. Networks 6 (1982).
10. Bachman, C. W. Implementation of techniques for data structure sets. In Proceedings of SHARE Workshop on DataBase Systems (Montreal, Canada, July, 1973).