Реляционная база данных: практическая основа эффективности

Э. Д. Кодд

Исследовательская лаборатория фирмы IBM в Сан-Хосе (IBM San Jose Research Laboratory)

Премия Тьюринга 1981 г. Ассоциации вычислительных машин (ACM) была вручена президентом Питером Деннингом 9 ноября 1981 г. Эдгару Ф. Кодду, сотруднику Исследовательской лаборатории фирмы IBM в Сан-Хосе, на ежегодной конференции ACM. Конференция состоялась в Лос-Анджелесе, Калифорния. Это главная премия Ассоциации, присуждаемая за выдающийся технический вклад в информатику.

Генеральный комитет ACM по премиям за технические достижения выбрал Кодда за его «продолжительный фундаментальный вклад в теорию и практику развития СУБД». Создатель реляционной модели баз данных, Кодд много сделал для развития реляционной алгебры (алгебры отношений), реляционного исчисления и нормализации отношений (normalization of relations).

Эдгар Кодд поступил на фирму IBM в 1949 г. Его задачей было писать программы для системы, называемой Selective Sequence Electronic Calculator. С тех пор его работа в области информатики была связана с логическим проектированием компьютеров (IBM701 и Stretch). Он руководил компьютерным центром в Канаде, возглавлял создание одной из первых операционных систем с возможностью мультипрограммного режима работы, занимался логикой самовоспроизводящихся автоматов, развивал методы высокого уровня для технических условий на средства программирования (softrware specifications), создавал и совершенствовал реляционный подход к управлению базами данных и работал над подсистемой анализа и синтеза английской речи для случайных пользователей баз данных. Он также является автором «Клеточных автоматов», ранней книги из серии монографий, издаваемых Ассоциацией вычислительных машин.

Кодд получил степени бакалавра и магистра искусств по математике в Оксфордском университете в Англии и степени магистра естественных наук и доктора философии в области вычислительной техники и информатики в Мичиганском университете. Он является членом Национальной академии инженерных наук США (National Academy of Engeneering (USA))

и Британского компьютерного общества (British Computer Society).

Премия Тьюринга ACM присуждается ежегодно в честь А. М. Тьюринга, английского математика, чьи исследования явились большим вкладом в информатику.

Хорошо известно, что увеличивающийся спрос конечных пользователей на новые приложения превосходит возможности отраслей, занимающихся обработкой информации, по созданию прикладных программ. Существует два взаимно дополняющих подхода к решению этой проблемы (и оба они являются необходимыми): один заключается в том, чтобы дать возможность конечным пользователям напрямую взаимодействовать с информацией, хранящейся в памяти компьютера; второй состоит в увеличении производительности труда профессионалов в области обработки данных при создании прикладных программ. Менее известно, что один-единственный метод — управление реляционными базами данных — дает практическую основу для обоих подходов. В данной статье объяснено, почему это так.

При обсуждении вопросов производительности отмечено, что пора провести четкую границу между реляционными и нереляционными базами данных, чтобы неправильное истолкование термина «реляционная» не вводило в заблуждение. Ключом к тому, где проводить эту границу, является нечто, называемое «возможностью работы с реляционной моделью данных».

1. ВВЕДЕНИЕ

Общеизвестно, что сейчас происходит «кризис производительности» в области создания пользовательских программ («running code») для коммерческих и промышленных применений. Отрасли, занимающиеся обработкой информации, уже неспособны обеспечить прикладными программами все увеличивающиеся потребности конечных пользователей в новых приложениях. В конце семидесятых — начале восьмидесятых годов многие люди, работающие в области информатики, надеялись, что введение систем управления базами данных (для их обозначения принято сокращение СУБД) значительно повысит эффективность работы прикладных программистов, устранив многочисленные проблемы, возникающие при работе с входными и выходными файлами. Оказалось, что СУБД (вместе со словарями данных) являются очень эффективным средством управления данными, и они действительно избавили программистов от множества проблем, возникающих при обработке файлов. Почему все же не удалось увеличить производительность?

Существуют три основные причины этого:

(1) Эти системы обременили прикладных программистов многочисленными понятиями, не имеющими прямого отношения к задачам поиска и обработки информации, вынуждая их думать и писать на неоправданно низком уровне структурных подробностей (наглядным примером этого являются «владель-

цы и члены множества»[1], которых придумали члены группы по созданию базы данных CODASYL);

(2) Не было предусмотрено средств для одновременной обработки нескольких записей — другими словами, СУБД не поддерживали работу с множествами, и в результате программисты были вынуждены думать и писать на языке циклов итераций, что часто было вовсе не обязательно (здесь мы используем слово «множество» в его традиционном математическом смысле, а не в смысле связной структуры, как подразумевает рабочая группа о базе данных CODASYL);

(3) Тот факт, что конечному пользователю потребуется непосредственное взаимодействие с базами данных, особенно непредсказуемое по своему характеру взаимодействие, был неправильно истолкован: считалось, что возможность работы с запросами — это то, что можно добавить к СУБД когда-нибудь потом.

Оглядываясь на системы управления базами данных, созданные в конце шестидесятых годов, можно сразу заметить, что в них не было существенного различия между представлением об информации программиста (логическим) и представлением информации в памяти машины (физическим). Если даже при так называемом логическом представлении защита от размещения осуществлялась на языке адресов памяти и побайтовых сдвигов, то и многие понятия, ориентированные на работу с памятью, были неотъемлемой частью того же уровня представлений.

Требование к программистам ориентироваться в путях доступа, чтобы достигнуть целевой информации (в некоторых случаях им приходилось иметь дело непосредственно с представлением данных в памяти, а в других — следовать по цепочке указателей) оказало очень неблагоприятное влияние. Ко всему прочему любое небольшое изменение размещения в памяти приводило к немедленной переделке всех программ, использующих предыдущую структуру. Введение индекса (предметного указателя) могло привести к тому же эффекту. В результате слишком много человеческих сил было истрачено на постоянную поддержку прикладных программ, которой можно было бы избежать.

---

[1] Причина трудностей, связанных с этой концепцией, состоит в том, что она объединяет в одной конструкции три независимых понятия: отношение «один — несколько», зависимость от существования объекта и видимая пользователю связная структура, вдоль которой должны пролагать свои маршруты прикладные программы. Именно последнее из этих понятий налагает на прикладных программистов тяжкое и ненужное бремя ориентирования в путях доступа, а также создает непреодолимые препятствия для конечных пользователей.

Другое следствие состояло в том, что ввод в эксплуатацию таких систем в большинстве случаев происходил очень медленно, потому что прежде чем активировать базу данных, много времени уходило на обучение работе в системе и на планирование организации данных как на логическом, так и на физическом уровне. Целью этого предварительного планирования было «сделать правильно сразу и навсегда», чтобы избежать необходимости впоследствии изменять описания данных, что в свою очередь повлечет изменения в прикладных программах. Такая цель, конечно, была недостижима, даже если бы основные принципы построения баз данных были известны в то время (а они, конечно, не были известны).

Чтобы показать, как системы управления реляционными базами данных избегают трех перечисленных выше ловушек, мы вначале рассмотрим причины, побудившие создавать реляционную модель данных, и обсудим некоторые ее черты. Потом мы займемся классификацией систем, основанных на этой модели. По мере продвижения мы будем обращать внимание на работу прикладного программиста, хотя выгоды для конечного пользователя также очень велики. О значении реляционных баз данных уже много говорилось и было приведено много примеров (см. [23] и ссылки в этой работе).

2. МОТИВИРОВКА

Основной побудительной причиной исследований, результатом которых стало создание реляционной модели данных, было желание четко разграничить логический и физический аспекты управления базами данных (принимая во внимание проблемы создания баз данных, поиска и обработки информации). Мы назвали это *стремлением к независимости данных.*

Вторым нашим желанием было создать структурно простую модель, так чтобы пользователи и программисты любой квалификации одинаково могли бы понимать содержащуюся в ней информацию и могли бы поэтому общаться друг с другом при помощи базы данных. Мы назвали это *стремлением к коммуникабельности.*

В-третьих, мы хотели использовать концепции языка высокого уровня (но не специфический синтаксис), чтобы пользователи имели возможность описывать операции сразу над большими порциями информации. Это является основой для способов обработки информации, ориентированных на множества (т. е. возможности при помощи одного оператора задать операцию над несколькими множествами записей одновременно). Мы назвали это *стремлением к обработке множеств.*

У нас также были и другие цели, такие как разработка тео-

ретических основ организации и управления базами данных, однако они не имеют непосредственного отношения к нашей теме эффективности.

3. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

Чтобы реализовать все три стремления, было необходимо отказаться от всех тех понятий структуризации данных (например, повторяющихся групп, связных структур), которые не известны конечным пользователям, и выработать новый взгляд на адресацию информации.

Понятие позиционирования всегда играло важную роль при работе с адресами в памяти компьютера, начиная со способа адресации с помощью наборного поля (коммутационной панели), потом абсолютной численной адресации, относительной численной адресации и символической адресации с арифметическими свойствами (как, например, символический адрес вида $A+3$ в ассемблере; адрес $X(I+1, J-2)$ элемента массива X в Алголе, Фортране, PL/I). В реляционной модели мы заменяем позиционный способ адресации общим ассоциативным способом.

Каждая единица информации в реляционной базе данных имеет уникальный адрес в виде имени отношения (relation name), значения первичного ключа данных и имени атрибута. Ассоциативный способ адресации такого вида позволяет пользователям (да-да, и даже программистам) предоставить системе самой (1) детально определять, где разместить новый фрагмент информации, вводимый в базу данных, и (2) выбирать соответствующие пути доступа при поиске данных.

Вся информация в реляционной базе данных представляется в виде значений в таблицах (даже имена таблиц являются символьными строками по крайней мере в одной таблице). Адресация информации по значению, а не по позиции, увеличивает эффективность работы как программистов, так и конечных пользователей (позиция элемента в последовательности обычно подвергается изменениям, и за ней довольно трудно следить, особенно если последовательность содержит много элементов). Более того, тот факт, что как программисты, так и конечные пользователи используют одинаковый способ адресации, дает возможность реализовать свойство коммуникабельности.

В качестве структурной единицы для реляционной модели данных было выбрано отношение n-го порядка, потому что при помощи соответствующих операторов и соответствующего концептуального представления (таблицы) оно ведет к реализации всех трех свойств, перечисленных выше. Заметим, что отноше-

ние n-го порядка — это математическое множество, в котором порядок строк не имеет значения.

Иногда возникают следующие вопросы: Почему такая модель называется реляционной моделью данных? Почему бы не назвать ее табличной? Для этого есть две причины: (1) Когда реляционная модель данных была предложена впервые, многие люди, занимающиеся обработкой данных, считали, что отношение между двумя или более объектами нужно представлять с помощью связных структур данных (и название было дано так, чтобы учесть это неправильное представление); (2) понятие таблицы находится на более низком уровне абстракции, чем понятие отношения, так как при его использовании создается впечатление, что применим способ адресации по позиции (в то время как это неверно для отношений n-го порядка), и тот факт, что содержание информации в таблице не зависит от порядка строк, не является очевидным. Тем не менее, даже обладая этими несущественными недостатками, таблицы являются основным средством концептуального представления отношений, потому что слово «таблица» понятно всем.

В тех случаях, когда какая-либо модель данных рассматривается как серьезная альтернатива реляционной модели, она также должна иметь четко определенное концептуальное представление для того, чтобы на ее основе создать базу данных. Наличие такого представления упрощает размышления о предполагаемых операциях. Это необходимо для эффективной работы программистов и конечных пользователей. В моделях данных, в которых используются такие понятия, как объекты (entities) и отношения, или в функциональных моделях данных такое представление если и обсуждается, то очень редко. В таких моделях часто вообще нет операторов! Тем не менее такие модели могут быть полезны при некоторых способах анализа типов данных, который приходится делать при создании новых баз данных, и на совсем ранних этапах работы, когда предварительно на неформальном уровне определяются способы ее организации. Все это приводит к вопросу: а что же это такое — модель данных?

Модель данных — это, конечно, не структура данных, как, возможно, думают многие. Естественно, что основные модели данных называются в соответствии с их основными структурами данных, но это еще не все.

Модель данных [9] — это комбинация по крайней мере трех составляющих:

(1) Набора типов структур данных (являющихся блоками при построении базы данных);

(2) Набора операторов или правил вывода, которые могут быть применены к любым правильным примерам типов дан-

ных, перечисленных в (1), чтобы находить, выводить или преобразовывать информацию, содержащуюся в любых частях этих структур в любых комбинациях.

(3) Набора общих правил целостности, которые прямо или косвенно определяют множество непротиворечивых состояний базы данных, или множество изменений ее состояния, или и то и другое. Эти правила являются общими в том смысле, что они применяются к любой базе данных, использующей данную модель. (Кстати, иногда они могут выражаться в виде правил ввода-обновления-вывода.)

Реляционная модель является моделью данных в этом смысле, и она является первой, удовлетворяющей этому определению. Мы не считаем нужным приводить в этой статье подробное определение реляционной модели — первое определение появилось в работе [7], затем оно было уточнено во второй и третьей частях работы [8]. *Структурная часть* реляционной модели данных состоит из доменов, отношений неопределенного порядка (relations of assorted degrees) (основным средством концептуального представления которых являются таблицы), атрибутов, кортежей (tuples), потенциальных ключей и первичных ключей. В соответствии с выбранным представлением атрибуты становятся столбцами таблиц, а кортежи — строками, но здесь, когда это касается таблиц нашей базы данных, не существует понятия того, что один столбец таблицы следует за другим или одна строка следует за другой. Другими словами, в этих таблицах порядок столбцов слева направо и порядок строк сверху вниз является произвольным и несущественным.

Обрабатывающая часть реляционной модели данных состоит из алгебраических операторов (выбора <select>, проекции <project>, соединения <join> и т. д.), которые преобразуют отношения в отношения (и, следовательно, таблицы в таблицы).

Часть, касающаяся *целостности*, состоит из двух правил — целостности объекта (entity integrity) и целостности на уровне ссылок (referential integrity) (последние достижения в этой области описаны в [8, 11]). В любой конкретной реализации модели данных может оказаться необходимым наложить дополнительные (зависящие от базы данных) ограничения, обеспечивающие целостность, и таким образом определить меньшее множество непротиворечивых (совместимых, значимых) состояний базы данных или их изменений.

В процессе развития реляционной модели данных всегда соблюдалось строгое соответствие между структурными аспектами и аспектами обработки и целостности. Если бы структуры создавались сами по себе, независимо от всего остального, свойства их поведения ничем не были бы связаны, и возникали бы бесконечные возможности и бесконечно продолжающиеся рас-

суждения. Поэтому неудивительно, что попытки, которые предпринимаются группами CODASYL и ANSI, развивать язык определения данных (DDL) и язык обработки данных (DML) в различных комитетах породили множество недоразумений и несовместимых результатов.

4. ВОЗМОЖНОСТЬ РЕЛЯЦИОННОЙ ОБРАБОТКИ ИНФОРМАЦИИ

Для реляционной модели данных необходимы не только реляционные структуры данных (которые можно считать таблицами), но также и определенный способ работы с множествами, который называется *реляционной обработкой данных.* При реляционной обработке данных приходится иметь дело с отношениями как с операндами. Ее основными задачами является предотвращение зацикливания (loop-aviodance), абсолютное требование эффективности для конечного пользователя и значительное увеличение производительности работы прикладных программистов.

Оператор реляционной алгебры SELECT (также иногда называемый RESTRICT) берет в качестве операнда *одно* отношение (таблицу) и преобразует его в новое отношение (таблицу), состоящее из выбранных кортежей (строк) первого отношения. Оператор PROJECT также преобразовывает *одно* отношение (таблицу) в новое, состоящее на этот раз из выбранных атрибутов (столбцов) первого. Оператор EQUI—JOIN берет в качестве операндов *два* отношения (две таблицы) и преобразует их в третью таблицу, строки которой состоят из строк первой таблицы, сцепленных со строками второй, но только в тех местах, где определенные столбцы первой и второй таблиц имеют совпадающие значения. Если исключаются избыточные столбцы, соответствующий оператор называется NATURAL JOIN. Впоследствии мы будем использовать термин «соединение» (join) для обозначения операторов соединения (equi-join) или естественного соединения (natural join).

Реляционная алгебра, включающая в себя эти и другие операторы, должна быть критерием производительности системы. Она не предназначена для того, чтобы быть стандартным языком, которого обязаны придерживаться все реляционные системы. Способность реляционной системы к обработке множеств необходимо поддерживать средствами подъязыка манипулирования данными [1], который по меньшей мере должен обладать

---

[1] Подъязык манипулирования данными — это специализированный язык для управления базой данных, поддерживающий по крайней мере операции определения (описания), поиска, ввода, обновления и удаления информации. Он не должен удовлетворять требованию вычислительной полноты и обычно

эффективностью реляционной алгебры, *не используя при этом итерационных и рекурсивных выражений.*

Мощность системы вывода, созданной на базе реляционной алгебры, определяется в основном только операторами SELECT, PROJECT и JOINT, при условии что оператор JOINT не подвергается таким ограничениям при реализации, при которых он должен иметь дело с предварительным определением соответствующих физических путей доступа. Система обладает *неограниченной способностью к соединению*, если она допускает соединения в *любой* паре сопоставимых атрибутов, при одном условии, что они определены на одном домене или типе данных (для наших целей неважно, является ли домен синтаксическим или семантическим и является ли тип данных слабым или сильным, однако в работе [10] описаны обстоятельства, при которых это существенно).

Иногда встречаются системы, в которых операция соединения выполняется только при условии, что сравниваемые атрибуты имеют одинаковое имя или поддерживаются заранее предопределенным путем доступа определенного типа. Такие ограничения значительно уменьшают способность системы к выводу новых отношений из основных. Следовательно, такие ограничения уменьшают возможности системы обрабатывать непредусмотренные запросы конечных пользователей и уменьшают шансы прикладного программиста избежать зацикливания.

Таким образом, мы говорим, что субъязык манипулирования данными L обладает *возможностью реляционной обработки*, если преобразования, определенные операторами реляционной алгебры SELECT, PROJECT и неограниченным оператором JOIN могут быть определены в L без использования итераций или рекурсии. Для того, чтобы система управления базой данных называлась *реляционной*, она должна поддерживать:

(1) Таблицы без видимых пользователю навигационных связей между ними;

(2) Подъязык манипулирования данными, обеспечивающий по крайней мере эту (минимальную) возможность реляционной обработки.

Из этого в первую очередь следует, что СУБД, которая *не* поддерживает реляционную обработку, должна рассматриваться как *нереляционная*. Более подходящим названием для такой системы является «*табличная*» при условии, что она поддерживает работу с таблицами без видимых пользователю навигационных путей между таблицами. Следовало бы заменить этим термином термин «полуреляционная», использованный

не удовлетворяет ему. С точки зрения программирования прикладных систем он предназначен для использования вместе с одним или несколькими языками программирования.

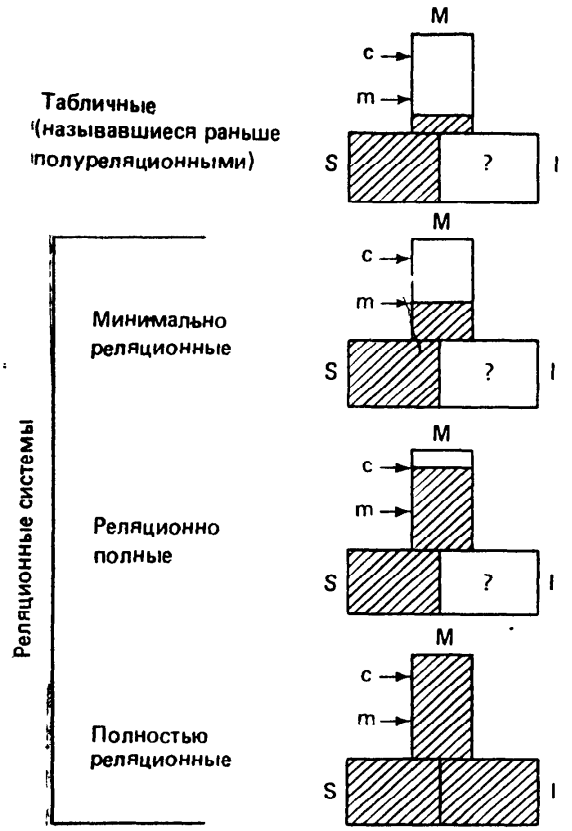

Рис. 1. Классификация СУБД: *S* — структурная часть, *M* — обрабатывающая часть; *I* — целостность, *c* — реляционная полнота, *m* — минимальная возможность реляционной обработки.

в работе [8], потому что существует большая разница в сложности реализации между табличными системами, в которых программист строит свои навигационные пути, и реляционными системами, которые самостоятельно строят пути доступа для программиста, т. е. системами, обеспечивающими *автоматическую навигацию*.

Данное выше определение реляционной СУБД допускает большую свободу при определении предоставляемых услуг. Например, вовсе не требуется, чтобы реляционная алгебра была реализована в полном объеме, и не предъявляется никаких требований, относящихся к поддержке двух правил целостности реляционной модели данных (целостности на уровне объектов и на уровне ссылок). Реляционная система, в которой в полной мере реализованы требования к двум последним частям реляционной модели данных, называется *полностью реляционной* [8]. Хотя в настоящее время не существует баз данных, которые можно было бы назвать полностью реляционными, многие системы уже очень близки к этому и нет сомнений, что очень скоро такие базы данных появятся.

На рис. 1 продемонстрированы различия между разными классами реляционных и табличных систем. Для каждого класса величина заштрихованной части прямоугольника **S** показывает, насколько в базах данных из этого класса выполняются структурные требования, предъявляемые к реляционной модели данных. То же самое относится и к прямоугольнику **M** и требованиям к обрабатывающей части базы данных, и к прямоугольнику **I** и требованиям к части базы данных, касающейся правил целостности.

Буква **m** соответствует минимальной возможности реляционной обработки, буква **c** означает реляционную полноту (т. е. таким возможностям системы, которые соответствуют логике

двуместных предикатов первого порядка без нулей. Если прямоугольник **М**, обозначающий обрабатывающую часть базы данных, заштрихован полностью, это значит, что в базе данных в полной мере реализованы возможности, соответствующие реляционной алгебре, определенной в [8] (возможности системы соответствуют логике трехместных предикатов с нулем одного типа). Для всех классов систем на рис. 1, кроме класса полностью реляционных баз данных, в прямоугольниках, обозначающих часть, реализующую правила целостности, стоит вопросительный знак. Это показывает, что в настоящее время в таких системах отсутствует адекватная реализация свойств целостности. Необходимы также более эффективная поддержка доменов и первичных ключей [10] и специальные средства реализации, описанные в работе [14].

Заметим, что реляционные СУБД могут сочетать возможности реляционной обработки любым удобным способом. Например, в системе INGRES фирмы Relational Technology, Inc. операция RETREIVE языка запросов QUEL [29] содержит все три оператора (выбора ⟨select⟩, проекции ⟨project⟩ и соединения ⟨join⟩) в одном запросе, так что можно получить тот же самый результат, что и при применении каждого из этих операторов по отдельности или любого их сочетания.

Определение реляционной модели данных содержит несколько запрещений. Приведем два примера: исключены видимые для пользователя навигационные связи между таблицами, и информация, содержащаяся в базе данных, не должна быть представлена (или скрыта) порядком кортежей в отношениях. Наш опыт говорит о том, что разработчики СУБД, которые имели дело с нереляционными системами, с трудом понимают и принимают эти ограничения. Пользователи же, наоборот, с энтузиазмом воспринимают значительное упрощение обучения и использования, которое является следствием этих запрещений.

Кстати, группа по реляционным системам Американского института стандартов (Relational Task Group of the American National Standards Institute) недавно опубликовала доклад о возможности создания стандарта для реляционных баз данных [4]. В этой работе содержится поучительный анализ доброй дюжины реляционных систем, и ее авторы хорошо понимают, что такое реляционная модель данных.

5. СВОЙСТВО УНИВЕРСАЛЬНОЙ РЕЛЯЦИОННОСТИ

Для расширения области применения в большинстве реляционных баз данных существуют подъязыки описания данных (манипулирования данными), которые могут иметь интерфейс

с одним или несколькими широко распространенными языками программирования (например, Коболом, Фортраном, PL/I, APL). Мы будем в дальнейшем называть последние *базовыми* языками. Реляционная СУБД обычно поддерживает по меньшей мере один подъязык описания данных, ориентированный на конечного пользователя, а иногда и несколько таких подъязыков, потому что потребности пользователей могут различаться. Одни предпочитают описательные языки (string languages), такие как QUEL или SQL [5], другим больше нравится экранный двумерный подъязык управления данными Query-by-Example (язык запросов на примерах) [33].

В настоящее время некоторые реляционные системы (например, System R [6], INGRES [29]) поддерживают языки описания данных, которые могут использоваться в двух режимах: (1) интерактивном, который необходим при работе с терминалом, и (2) в режиме «встроенности» в прикладную программу, написанную на базовом языке. Существуют серьезные причины возникновения подъязыков описания данных с *двумя режимами:*

(1) С помощью такого языка прикладной программист может независимо отлаживать с терминала только обращения к базе данных, которые он предполагает включить в свои программы. Программисты, использовавшие при работе SQL, считают, что наличие двух режимов значительно повышает эффективность их работы (скорость отладки программ);

(2) Язык такого типа значительно увеличивает коммуникационные возможности СУБД для программистов, аналитиков, конечных пользователей, персонала администрации базы данных и т. д.;

(3) Произвольные различия между разными языками, используемыми в двух режимах, излишне обременяют память тех пользователей, которым приходится работать в обоих режимах, и требуют дополнительных усилий при обучении.

Наличие двух режимов в языке описания данных является настолько важным для эффективности системы, что существует классификация реляционных СУБД в зависимости от того, обладают они этим свойством или нет. Мы называем реляционные базы данных, в которых поддерживается подъязык описания данных с двумя режимами, *универсально реляционными.* Таким образом, универсально реляционные СУБД поддерживают реляционную обработку как на уровне интерфейса с конечным пользователем, так и на уровне интерфейса с прикладным программистом *с помощью подъязыка описания данных, общего для обоих интерфейсов.*

Естественным названием для всех остальных СУБД является *«неуниверсально реляционные».* Примером неуниверсально-

реляционной базы данных является TANDEM ENCOMPASS [19]. В этой системе при интерактивном режиме поиска информации с терминала используется реляционный подъязык описания данных ENFORM (это язык, обладающий возможностями реляционной обработки). При написании программы, в которой требуется найти или обработать информацию, используется расширенная версия языка Кобол (языка, который не обладает возможностями для реляционной обработки). На обоих уровнях используются общие структуры: таблицы без видимых пользователю навигационных путей между ними.

Немедленно возникает следующий вопрос: каким образом подъязык, обладающий возможностями реляционной обработки, сопрягается с такими языками, как, например, Кобол или PL/I, которые могут обрабатывать записи только по одной (т. е. не обладают возможностью воспринимать множество записей как один операнд). Чтобы решить эту проблему, необходимо различать два действия: (1) определение отношения, которое выводится; (2) представление выведенного отношения в программе, написанной на базовом языке.

Одно решение (принятое в системе Peterlee Relational Test Vehicle [31]) состоит в том, чтобы представлять выведенное отношение в виде файла, который может быть прочитан запись за записью с помощью операторов базового языка. В этом случае записи выдаются в файл той файловой системы, которую используют конкретный базовый язык.

Другое решение (принятое в системе System R) состоит в том, чтобы управлять выдачей записей с помощью предложений подъязыка описания данных и, следовательно, оптимизатора реляционной СУБД. Оператор запроса Q языка SQL (подъязыка описания данных системы System R) может быть помещен в программу, написанную на базовом языке, в виде следующего предложения (для наглядности его синтаксис не совсем точно совпадает с синтаксисом SQL):

DECLARE C CURSOR FOR Q

где C — любое имя, выбранное программистом. Такое предложение ставит в соответствие (ассоциирует) курсор с именем C и определение выражения Q. Кортежи из выведенного отношения, определенного запросом Q, предъявляются программе по одному с помощью курсора с именем. Каждый раз, когда посредством этого курсора выполняется операция выборки FETCH, система выдает следующий кортеж выведенного отношения. Порядок выдачи определяется системой, за исключением тех случаев, когда запрос Q, определяющий выведенное отношение, содержит условие ORDER BY.

Важно отметить, что продвигая курсор по выведенному от-

ношению, программист не участвует в процессе навигации по направлению к некоторой целевой информации. Полученное отношение само и является целевой информацией! Именно СУБД сама определяет, должно ли выведенное отношение быть материализовано целиком перед тем, как просматривать его с помощью курсора, или по частям в процессе просмотра. В любом случае именно система (а не программист) выбирает пути доступа, при помощи которых генерируются полученные данные. Это снимает с плеч программиста огромный груз, повышая таким образом эффективность его работы.

6. СКЕПТИЧЕСКОЕ ОТНОШЕНИЕ К РЕЛЯЦИОННЫМ СИСТЕМАМ

Не было недостатка в скептических замечаниях, касающихся целесообразности реляционного подхода к управлению базами данных. Непонимание было источником большей их части, источником других был страх перед многочисленными теоретическими исследованиями, основанными на реляционной модели [1, 2, 15, 16, 24]. Вместо того, чтобы приветствовать создание теоретических основ, обеспечивающих глубину проработки, позиция большинства, кажется, такова: что хорошо в теории, будет плохо на практике. Почти для всех нереляционных СУБД не существует теоретического обоснования, что и является основной причиной того, что их можно назвать ungepotchket (это слово из языка идиш, одним из значений которого является «составленный из лоскутков или обрезков»).

С другой стороны, кажутся разумными следующие два вопроса:

(1) Может ли реляционная система обеспечить такой же сервис, какой мы привыкли ожидать от других СУБД?

(2) Если да, то может ли такая система работать не менее эффективно, чем нереляционная СУБД?[1)]

Мы рассмотрим по очереди каждый из этих вопросов.

6.1. УРОВЕНЬ СЕРВИСА

Полноценная СУБД предоставляет следующие возможности:

* хранение, поиск и обновление данных;
* доступный пользователю каталог для описания информации;
* поддержка транзакций, чтобы иметь уверенность в том,

[1)] Следует помнить, что в нереляционных системах для прикладного программирования всегда используются подъязыки описания данных более низкого уровня, чем в реляционных.

что если в базу данных вводится последовательность изменений, то все они или ни одно из них правильно отражены в текущем состоянии базы данных;

* возможности восстановления в случае ошибки или отказа (системы, носителей информации или программы);

* средства управления параллельным выполнением операций, для того, чтобы одновременная обработка транзакций давала тот же результат, что и последовательное их выполнение в некотором порядке;

* поддержка средств контроля доступа (санкционирования доступа, проверки полномочий), чтобы обеспечить соответствие доступа и обработки информации специфическим ограничениям, наложенным на пользователя или программу;

* средства объединения (интеграции) и поддержки обмена информацией;

* средства обеспечения целостности, чтобы обеспечить соответствие состояния базы данных и изменений этого состояния определенным правилам.

Немногочисленные прототипы реляционных СУБД, созданные в начале 70-х годов, были очень далеки от предоставления всех перечисленных возможностей (вероятно, по очень уважительным причинам). Однако сейчас некоторые реляционные системы, имеющиеся на рынке программного обеспечения, обладают всеми этими возможностями, за исключением последней. Современные версии этих систем, по общему признанию, не обеспечивают в достаточной мере целостность информации, но эта ситуация быстро исправляется [10].

В настоящее время некоторые реляционные СУБД предоставляют более полный набор услуг по обработке данных, чем нереляционные системы. Приведем три примера.

Во-первых, реляционная СУБД обеспечивает выделение всех значимых отношений, существующих в базе данных, в то время как нереляционные системы находят информацию только в тех случаях, когда существуют статически предопределенные пути доступа.

Во-вторых, в качестве примера дополнительных услуг, предоставляемых реляционными системами, рассмотрим разрезы данных (views). *Разрез данных* — это виртуальное отношение (таблица), которая является результатом обработки выражения или последовательности команд. Хотя разрез данных на самом деле не поддерживается непосредственно в текущем состоянии базы данных, он может быть предъявлен пользователю, как если бы он в настоящий момент существовал в базе данных как дополнительная базовая таблица, состояние которой согласовано с состоянием остальных базовых таблиц. Разрезы полезны, так как они позволяют прикладным программистам и пользовате-

лям, сидящим за терминалами, взаимодействовать с постоянными структурами разрезов даже в то время, когда сами базовые таблицы подвергаются структурным изменениям на логическом уровне (при условии, что соответствующие разрезы определимы на основе обновленных базовых таблиц). Они также полезны для организации ограничения уровня доступа программ и пользователей. Нереляционные системы или совсем не поддерживают работу с разрезами данных, либо поддерживают гораздо более примитивные эквиваленты, как, например, подсхема в СУБД CODASYL.

И в-третьих, некоторые системы (например, SQL/DS [28] и ее более ранний прототип System R) позволяют делать различные изменения в логической и физической структуре данных в динамическом режиме — одновременно с обработкой транзакций. Обычно внесение таких изменений требует переделки прикладных программ. Таким образом, это свойство уменьшает количество работы, приходящееся на поддержание работоспособности программ, тем самым повышая эффективность труда программистов и давая им возможность заниматься усовершенствованием своих программ, а не их поддержкой. Эту возможность удалось реализовать в системе SQL/DS благодаря тому, что система сама полностью контролирует выбор путей доступа.

В нереляционных системах для внесения таких изменений обычно требуется остановка всей остальной деятельности СУБД, в том числе должно быть прервано и выполнение обрабатываемых транзакций. База данных остается блокированной до тех пор, пока не закончится выполнение организационных изменений, и не будет выполнена компиляция.

6.2. ЭФФЕКТИВНОСТЬ

Конечно, люди не будут использовать реляционные системы, если эти системы будут работать медленно. Но все же слишком часто делают ошибочные выводы об эффективности реляционных систем, сравнивая время, которое могло бы потребоваться одной из таких систем для обработки сложной транзакции, с временем, которое потребовалось бы нереляционной системе для выполнения очень простой транзакции. Для того чтобы произвести честное сравнение, нужно сравнивать эти системы на одних и тех же задачах или применениях. Мы представим аргументы, которые покажут, почему реляционные системы должны быть способны соревноваться на равных с нереляционными системами.

Хорошая эффективность определяется двумя факторами: (1) система должна поддерживать физические структуры дан-

ных, ориентированные на увеличение эффективности; (2) запросы к данным, написанные на языке высокого уровня, должны компилироваться в коды более низкого уровня с такой же эффективностью, как это может сделать вручную средний прикладной программист.

Первым аргументом в нашем споре будет тот факт, что программу, написанную на языке того же уровня, что и Кобол, можно эффективно исполнять в больших базах данных, обладающих средствами для вывода данных, структурированных в виде таблиц без видимых пользователю навигационных связей между ними. В поддержку этого аргумента приведем следующую информацию: к августу 1981 г. корпорацией Tandem Computer Corp. было инсталлировано 760 систем, в более чем 700 из них для поддержки баз данных, содержащих сведения о продукции, использовалась реляционная система управления базой данных Tandem ENCOMPASS. Корпорация Tandem доверила свою собственную базу данных, содержащую сведения о производстве, заботам системы ENCOMPASS. Система ENCOMPASS не поддерживает ни видимые пользователю (навигационные) связи, ни невидимые для пользователя связи (пути доступа).

В качестве второго аргумента в споре произведем следующие действия. Возьмем прикладные программы, работающие с описанной выше инсталлированной системой, и запишем все операции поиска и обработки данных с помощью операторов подъязыка описания данных, обладающего возможностями реляционной обработки (т. е. SQL). Ясно, что при использовании языка высокого уровня такого типа можно добиться высокой эффективности, только если он компилируется в объектный код (программу на языке транслятора), а не интерпретируется, и этот объектный код достаточно эффективен.

Компиляция использовалась в системе System R и ее коммерческой версии SQL/DS. В 1976 г. Раймонд Лорие придумал остроумную схему пред- и посткомпиляции, чтобы работать с динамическими изменениями в путях доступа. С ее помощью также на ранних этапах (и поэтому эффективно) определяются ограничения на доступ и проверяется целостность (последнее, однако, еще не реализовано). При использовании этой схемы компиляция выражений на подъязыке SQL, включенных в прикладную программу, написанную на базовом языке, происходит довольно своеобразно. Шаг компиляции преобразует выражение на языке SQL в соответствующие вызовы (CALLs) в исходной программе и в модули доступа, содержащие объектный код. Эти модули хранятся в базе данных, чтобы потом использовать их в процессе прохождения программы. Код, содержащийся в модулях доступа, генерируется системой так, чтобы

оптимизировать последовательность выполнения основных операций и обеспечить эффективную работу программы. После выполнения такой предкомпиляции прикладная программа компилируется обычным компилятором базового языка. Если впоследствии один или несколько путей доступа будут удалены и будет сделана попытка запустить программу, в модуле доступа останется достаточно информации для того, чтобы система могла скомпилировать новый модуль доступа, содержащий существующие на текущий момент пути доступа, и для этого *не потребуется снова компилировать прикладную программу.*

Кстати, тот же самый компилятор с подъязыка описания данных используется для обработки непредвиденных запросов (ad hoc queries), поступающих интерактивно с терминала, а также запросов, динамически формируемых в процессе исполнения программы (т. е. с помощью интерактивно введенных параметров). Такие запросы выполняются сразу после компиляции, и, за исключением простейших из них, эффективность выше, чем при использовании интерпретатора.

Генерация модулей доступа (как на стадиях начальной компиляции, так и рекомпиляции) влечет за собой применение довольно сложной схемы оптимизации [27], в которой используются накопленные системой статистические данные, которые в нормальной ситуации неизвестны программисту. Таким образом, средний прикладной программист мог бы соревноваться с этим оптимизатором в создании эффективного кода только для простейшей из транзакций. Любые попытки соревноваться связаны с уменьшением эффективности работы программиста. Таким образом, цена, которую приходится платить за издержки, связанные с увеличением времени компиляции, оказывается вполне приемлемой.

Предполагая, что в обоих случаях используются несвязанные табличные структуры, можно ожидать, что система SQL/DS будет генерировать код, сравнимый по эффективности с кодом, написанным вручную программистом средней квалификации, и превосходящий его по эффективности в сложных случаях. Многие транзакции, используемые в коммерции, крайне просты. Например, может потребоваться найти запись об определенном железнодорожном вагоне, чтобы узнать, где он находится, или подсчитать чьи-либо сбережения. Если поддерживаются достаточно быстрые пути доступа (например, хэшинг), непонятно, почему такие языки высокого уровня, как SQL, QUEL и QBE, должны вырабатывать для этих простых транзакций менее эффективный код, чем язык более низкого уровня, даже если при обработке этих транзакций не используются оптимизирующие возможности компилятора языка высокого уровня.

7. НАПРАВЛЕНИЯ ДАЛЬНЕЙШЕЙ РАБОТЫ

Если мы собираемся использовать реляционную базу данных для повышения эффективности, нам необходимо знать, какие усовершенствования можно внести в реляционную систему.

Сначала давайте поговорим об изменениях, осуществимых в ближайшее время. В некоторых реляционных системах необходима более сильная поддержка доменов и первичных ключей, как это предлагается в [10]. Уже было отмечено, что все реляционные системы нуждаются в наращивании вычислительных возможностей, чтобы ограничения, связанные с обеспечением целостности, удовлетворялись автоматически. Существующие ограничения на обновление разрезов данных, полученных в результате операции соединения над отношениями, должны стать менее строгими (где это теоретически возможно), и в решении этой проблемы достигнут значительный прогресс [20]. Необходимо также создавать средства поддержки операции соединения над внешними отношениями.

Методы оптимизации значительно совершенствуются, так что есть основания ожидать дальнейшего увеличения эффективности. В некоторых коммерческих системах, таких как ICL CAFS [22] и в системе IDM500 компании Britton-Lee [13], были созданы специальные средства аппаратной поддержки. В некоторых приложениях эффективность можно увеличивать за счет использования специальных технических средств. Однако в большинстве применений, использующих базы данных, реализованные в виде программ, реляционные базы данных могут соревноваться в эффективности с реализованными в виде программ нереляционными системами.

В настоящее время большинство реляционных систем не обеспечивают специальных средств для работы с инженерными и научными базами данных. Однако необходимость в создании таких средств, в том числе интерфейса с Фортраном, очевидна.

Каталоги в реляционных системах уже состоят из дополнительных отношений, и их можно опрашивать так же, как и остальную информацию в базе данных, используя тот же язык запросов. Естественным усовершенствованием, которое может и должно быть сделано в ближайшее время, является превращение этих каталогов в полностью оснащенные активные управляющие словари для дополнительного текущего контроля информации.

Наконец, в настоящее время можно ожидать появления средств создания баз данных, совместимых с реляционными системами как на физическом, так и на логическом уровнях.

Через более продолжительное время можно предсказать появление баз данных, распределенных по вычислительным сетям

[25, 30, 32] и управляемых таким образом, чтобы прикладные программы и пользователи, работающие в интерактивном режиме, могли работать с информацией так, что (1) как будто она вся хранится в локальном узле — это свойство называется *прозрачностью расположения* ⟨location transparency⟩ (независимость от местоположения, делающая его незаметным для пользователя); и (2) как будто информация нигде не дублируется — это свойство называется *прозрачностью дублирования* ⟨replication transparency⟩ (независимость от дублирования, делающая его незаметным для пользователя).

Все три перечисленных выше проекта основываются на реляционной модели. Свойство реляционности является важным, потому что реляционная база данных обладает большой гибкостью при декомпозиции и распределении ее модулей по сети вычислительных систем и широкими возможностями для динамического объединения распределенной информации. В противоположность этому, базы данных CODASYL DBTG очень трудно разделить на составные части и объединить вновь из-за путаницы навигационных связей между владельцами и членами множеств. Из-за этого подход, реализованный в базе данных CODASYL, очень трудно приспособить к работе в распределенной среде, и это может послужить причиной того, что от него придется отказаться. Вторая причина использования реляционной модели при работе в сети вычислительных систем заключается в том, что для такой модели созданы краткие подъязыки описания данных, удобные для передачи запросов между узлами сети.

Можно ожидать, что продолжающаяся работа по развитию возможностей реляционной модели лучше выражать смысл (значение) информации на формальном языке приведет к включению этого смысла в каталог базы данных, чтобы удалить его за рамки прикладных программ и тем самым сделать эти программы еще короче и проще. Здесь, конечно, мы говорим о смысле, который представляется таким образом, что система может его понимать и производить действия над ним.

Развиваются теоретические исследования, касающиеся обработки недостающей или неприменимой информации (как, например, в работе [3]). Результатом этих исследований будут улучшенные методы обработки неопределенных величин.

В настоящее время реляционные базы данных лучше приспособлены для работы с информацией с регулярной или однородной структурой. Сохранятся ли преимущества реляционного подхода при работе с неоднородными данными? Такие данные могут содержать изображения, тексты и разнообразные факты. Ожидается, что на этот вопрос будет дан положительный ответ,

исследования на эту тему развиваются, но их, конечно, еще недостаточно.

Серьезные исследования необходимы для сближения и достижения совместимости языков баз данных и языков программирования. Хорошим примером работы в этом направлении является Pascal/R [26]. Целью продолжающихся исследований является, с одной стороны, включение абстрактных типов данных в языки баз данных [12], и с другой стороны, создание средств реляционной обработки в языках программирования.

8. ВЫВОДЫ

Мы представили множество аргументов в поддержку мнения, что технология реляционных баз данных предоставляет возможности для впечатляющего увеличения эффективности как конечным пользователям, так и прикладным программистам. Основными доводами являются такие свойства реляционных систем, как независимость данных, структурная простота и возможности реляционной обработки. Эти свойства были определены для реляционной модели данных и реализованы в виде реляционных систем управления базами данных. Все три перечисленных свойства упрощают создание прикладных программ и формулировку запросов и изменений информации, производимых с терминала. Кроме того, первое свойство делает программы устойчивыми к изменениям описаний и структур в базе данных и тем самым уменьшает трудозатраты на поддержание работоспособности программ.

Почему же тогда в названии лекции утверждается, что реляционная база данных только дает основания для повышения эффективности, а не решает эту задачу полностью? Ответ простой: реляционные базы данных имеют дело только с проблемами взаимодействия с пользователями и с теми компонентами прикладных программ, которые касаются совместно используемых данных. Существует множество других технологий, которые относятся к другим компонентам и аспектам создания баз данных, например, разработка языков программирования, в которых поддерживаются реляционная обработка и методы контроля типов данных, создание редакторов, «понимающих» используемый язык программирования и т. д. Мы использовали слово «основания», потому что взаимодействие с распределенной информацией (на уровне программы или через терминал) является основой большей части деятельности по обработке данных.

Практичность реляционного подхода доказана многочисленными тестами и инсталляцией коммерческих систем, которая уже происходит. Поэтому по мере развития реляционных систем мы можем ожидать резкого увеличения эффективности, которое,

как мы надеялись, появление СУБД обеспечит в первую очередь.

БЛАГОДАРНОСТИ

Я хотел бы выразить свою признательность членам группы по разработке системы System R из исследовательской лаборатории фирмы IBM в Сан-Хосе за то, что они создали полноценный прототип базы данных, обладающей свойством универсальной реляционности, что привело к изобретению многочисленных нововведений, касающихся языка и организации системы. Я благодарен также разработчикам из лаборатории фирмы IBM в Эндикотте, штат Нью-Йорк, за профессионализм, с которым они превратили систему System R в коммерческий продукт, а также многочисленным сотрудникам университетов, создателям аппаратных средств, фирмам, создающим программное обеспечение и пользователям, которые придумали и реализовали работающие реляционные системы. Я благодарен группе QBE из лаборатории IBM Yorktown Heights, штат Нью-Йорк, группе PRTV научного центра фирмы IBM в Англии и многочисленным ученым, занимавшимся теорией баз данных, для которых реляционная модель была краеугольным камнем. Специальную благодарность я приношу тем немногочисленным коллегам, которые увидели что-то стоящее в моих исследованиях и поддержали меня на ранних этапах работы, особенно Крису Дейту и Шарон Вейнберг. В конце концов именно Шарон Вейнберг придумала тему этой статьи.

ЛИТЕРАТУРА

1. Berri C., Bernstein P., Goodman N. A sophisticate's introduction to database normalization theory. *Proc. Very Large Data Bases,* West Berlin, Germany, Sept. 1978.
2. Bernstein P. A., Goodman N., Lai M.-Y. Laying phantoms to rest. Report TR-03-81, Center for Research in Computing Technology, Harvard University, Cambridge, Mass., 1981.
3. Biscup J. A. A formal approach to null values in database relations. *Proc. Workshop on Formal Bases for Data Bases,* Toulouse, France, Dec. 1979; опубликовано в [16] (см. ниже), pp. 299—342.
4. Brodie M., Schmidt J. (Eds.) Report of the ANSI Relational Task Group.
5. Chamberlin D. D. et al. SEQUEL2: A unified approach to data definition, manipulation, and control. *IBM J. Res. & Dev. 20,* 6 (Nov. 1976), 560—565.
6. Chamberlin D. D. et al. A history and evaluation of system R. *Comm. ACM 24,* 10 (Oct. 1981), 632—646.
7. Codd E. F. A relational model of data for large shared data banks. *Comm. ACM* **13,** 6 (June 1970), 377—387.
8. Codd E. F. Extending the database relational model to capture more meaning. *ACM TODS 4,* 4 (Dec. 1979), 397—434.
9. Codd E. F. Data models in database management. *ACM SIGMOD Record 11,* 2 (Feb. 1981), 112—114.

10. Codd E. F. The capabilities of relational database management systems. *Proc. Convencio Informatica Llatina,* Barcelona, Spain, June 9—12, 1981, pp. 13—26.
11. Date C. J. Referential integrity. *Proc. Very Large Data Bases,* Cannes, France, Sept. 9—11, 1981, pp. 1—12.
12. Ehrig H., Weber H. Algebraic specification schemes for data base systems. *Proc. Very Large Data Bases,* West Berlin, Germany, Sept. 13—15, 1978, pp. 427—440.
13. Epstein R., Hawthorne P. Design decisions for the intelligent database machine. *Proc. NNC 1980, AFIPS, Vol. 49,* May 1980, pp. 237—241.
14. Eswaran K. P., Chamberlin D. D. Functional specifications of a subsystem for database integrity. *Proc. Very Large Data Bases,* Framingham, Mass., Sept. 1975, pp. 48—68.
15. Fagin R., Horn clauses and database dependencies. *Proc. 1980 ACM SIGACT Symp. on Theory of Computing,* Los Angeles, CA, pp. 123—134.
16. Gallaire H., Minker G., Nicolas J. M. *Advances in Data Base Theory.* Vol. 1, Plenum Press, New York, 1981.
17. Gray J. The transaction concept: Virtues and limitations. *Proc. Very Large Data Bases,* Cannes, France, Sept. 9—11, 1981, pp. 144—154.
18. Griffiths P. G., Wade B. W. An authorisation mechanism for a relational database cyctem. *ACM TODS 1,* 3 (Sept. 1976), 242—255.
19. Held G. ENCOMPASS: A relational data manager. Data Base/81, Western Institute of Computer Science, Univ. of Santa Clara, Santa Clara, Calif., Aug. 24—28, 1981.
20. Keller A. M. Updates to relational databases through views involving joins. Report RJ3282, IBM Research Laboratory, San Jose, Calif., October 27, 1981.
21. Lorie R. A., Nilsson J. F. An access specification language for a relational data base system. *IBM J. Res. & Dev. 23,* 3 (May, 1979), 286—298.
22. Maller V. A. J. The content addressable file store—CAFS. *ICL Technical J. 1,* 3 (Nov. 1979), 265—279.
23. Reisner P. Human factors studies of database query languages: A survey and assessment. *ACM Computing Surveys 13,* 1 (March 1981), 13—31.
24. Rissanen J. Theory of relations for databases—A tutorial survey. *Proc. Symp. on Mathematical Foundations of Computer Science,* Zakopane, Poland, September 1978, Lecture Notes in Computer Science, No. 64, Springer Verlag, New York, 1978.
25. Rothnie J. B., Jr., et al. Introduction to a system for distributed databases (SDD-1). *ACM TODS 5,* 1 (March 1980), 1—17.
26. Schmidt J. W. Some high level language constructs for data of type relation. *ACM TODS 2,* 3 (Sept. 1977), 247—261.
27. Selinger P. G., et al. Access path selection in a relational database system. *Proc. 1979 ACM SIGMOD International Conference on Management of Data,* Boston, Mass., May 1979, pp. 23—34.
28. ——— SQL/Data system for VSE; A relational data system for application development. IBM Corp. Data Processing Division, White Plains, N. Y., Feb. 1981.
29. Stonebraker M. R., et al. The design and implementation of INGRES. *ACM TODS 1,* 3 (Sept. 1976), 189—222.
30. Stonebraker M. R., Neuhold E. J. A distributed data base version of INGRES. *Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks,* Lawrence-Berkeley Lab., Berkeley, Calif., May 1977, pp. 19—36.
31. Todd S. J. The Peterlee relational test vehicle — A system overview. *IBM Systems J. 15,* 4 (1976), 285—308.
32. Wiliams R. et al. R*: An overview of the architecture. Report RJ 3325, IBM Research Laboratory, San Jose, Calif., Oct. 27, 1981.
33. Zloof M. M. Query by example. *Proc. NCC, AFIPS, Vol. 44,* May 1975, pp. 431—438.

ПОСТСКРИПТУМ

Э. Ф. Кодд
Codd and Date Consulting Group
San Jose, Calif.

Подготавливая доклад по случаю получения премии Тьюринга, я преследовал две цели: (1) подчеркнуть, что реляционная модель определяет не только структурные аспекты организации данных, что хорошо заметно пользователям — она также определяет и способы обработки и аспекты, касающиеся целостности информации; и (2) выделить минимальный набор свойств реляционной модели, которые можно было бы использовать, чтобы отличить реляционные системы управления базами данных (СУБД) от нереляционных систем. Для того, чтобы система могла называться реляционной, необходимо, чтобы она поддерживала каждое из этих свойств.

В первой половине 1980-х годов большинство фирм, продающих программное обеспечение, сообщили о создании коммерческих систем, которые они называли реляционными. Многие из этих фирм считали свои системы «полностью реляционными», в то время как их продукция удовлетворяла только минимальному набору качеств, исходя из которого эти системы можно было назвать реляционными. Некоторые фирмы выпустили на рынок системы, которые не удовлетворяли даже этим минимальным требованиям, но громко именовались в руководствах, рекламных объявлениях, на презентациях и в сообщениях для печати «полностью реляционными».

Чтобы защитить пользователей, которые рассчитывают получить все преимущества, вытекающие из реляционного подхода, осенью 1985 г. я решил опубликовать статью, состоящую из двух частей, «Насколько реляционной является Ваша система управления базой данных?» в журнале Computerworld (14 и 21 октября). В первой части этой статьи я описал 12 свойств, каждое из которых коммерческая СУБД должна реализовывать в полной мере, чтобы у нее были шансы действительно считаться реляционной. Во второй части я оценил три широко разрекламированных коммерческих СУБД, рассчитанных на широкое применение. Две из этих систем получили нулевые оценки за реализацию этих 12 свойств.

Я верю, что эти статьи оказали благотворное влияние и уменьшили количество пышных фраз, употребляемых фирмами, продающими реляционные базы данных.