История и актуальные проблемы темпоральных баз данных

Б.Б. Костенко, С.Д. Кузнецов

1. Введение

Целью данной статьи является обеспечение знакомства с темпоральными[1] базами данных, текущим состоянием разработок, а также актуальными задачами и дальнейшими путями развития данного направления[2]. Хочется подчеркнуть важность исследований в данной области, поскольку почти все данные, так или иначе, связаны с различными событиями, интервалами времени. Однако лишь незначительная часть разработчиков осознает, что это отдельная и вполне самостоятельная область исследований. Зачастую многие темпоральные системы создаются лишь на основе общих методов проектирования и разработки приложений баз данных, без использования накопленных знаний в области создания систем управления темпоральными базами данных. Подобное «изобретение велосипеда» не только повышает стоимость разработки, но и может самым негативным образом сказаться на эффективности работы приложений и наличии в них ошибок.

Что же такое темпоральные данные? В очень широком смысле – это произвольные данные, которые явно или неявно связаны с определенными датами или промежутками времени. Под такое определение попадают почти любые данные и информация. Например, даже если нет явной зависимости от времени для какой-нибудь аксиомы, то все равно для нее имеется неявная

[1] В русскоязычной литературе вместо термина «темпоральные базы данных» иногда применяется термин «временны*ы*е базы данных». Но поскольку, во-первых, в этой области отсутствует устоявшаяся русскоязычная терминология и, во-вторых, при использовании первого термина, являющегося калькой английского термина *temporal database*, не требуется дополнительное уточнение ударения, в этой статье будет использоваться именно этот термин.

[2] За исключением специально оговоренных случаев в этой статье речь будет идти о реляционной модели данных (и ее темпоральных расширениях). Кроме того, здесь не проводится различия между реляционной моделью данных в классическом смысле и моделью данных языка SQL, и используются термины, принятые в мире SQL-ориентированных баз данных.

зависимость от времени, так как когда-то нам (или системе) стало известно, что такая аксиома существует. Кроме того, есть вероятность, что в будущем аксиома будет опровергнута, или на условия ее применимости будут наложены определенные ограничения; поэтому нельзя уже будет рассматривать ее как некоторую абсолютную истину, верную во всех ситуациях и в любой момент времени[3].

Темпоральные базы данных – это базы данных, хранящие темпоральные данные. Однако эти базы данных и содержащиеся в них данные могут рассматриваться как темпоральные только в том случае, если известно правило интерпретации временных меток и интервалов для конкретной системы управления базами данных (СУБД). Чтобы определить, является ли рассматриваемая СУБД темпоральной в полном смысле этого слова, необходимо понять, можно ли отдельно выделить и специальным образом интерпретировать данные атрибута «время». В категорию темпоральных СУБД не будут попадать обычные реляционные СУБД, в которых поддерживаются связанные со временем типы данных, но интерпретацией и связью данных (или событий) между собой с учетом времени приходится заниматься разработчику. В «настоящей» темпоральной СУБД учитываются специфическая природа времени и изменчивость данных с течением времени.

Специальные публикации на русском языке, посвященные тематике темпоральных баз данных, почти полностью отсутствуют. В данной обзорной статье невозможно сколько-нибудь полно охватить все направления исследований нескольких последних десятилетий, поэтому в ней рассматриваются только ключевые этапы развития данной области исследований, а также наиболее актуальные исследования и разработки, известные в настоящее время.

В начале статьи кратко представлена история области исследований темпоральных баз данных и перечислены наиболее значимые поворотные моменты. Затем излагаются основные понятия и термины, идеи и подходы,

[3] Если пойти еще дальше, то можно даже сказать, что результаты любых вычислений, по сути, тоже являются темпоральными данными, так как связаны со временем. Например, представим, что определенное решение принимается на основе целочисленного округления значения некоторого вещественного выражения. Если система некоторое время округляет 0.5 до 0, а потом – до 1, то мы можем получить разные результаты на одних и тех же исходных данных, то есть $F(C) \neq F(C)$, где C – константа. Для формально корректной записи подобного неравенства требуется введение дополнительного аргумента – момента вычисления, и тогда получится абсолютно корректное выражение $F(C, t_1) \neq F(C, t_2)$. Данный пример может показаться несколько искусственным, но он демонстрирует, что время является неотъемлемым атрибутом любых данных, когда речь идет о практической работе с конкретной системой, а не лишь о теоретическом ее исследовании.

применяемые в рассматриваемой области, а также объясняются мотивы, побудившие начать исследования в данном направлении. Далее представлены наиболее важные расширения, дополнения и ответвления в области исследований темпоральных баз данных. Завершающая часть статьи посвящена направлениям и состоянию исследовательских работ в области темпоральных баз данных, выполняемых в настоящее время.

2. Краткая история

История систем реляционных баз данных насчитывает более трех десятилетий. Примерно на 10 лет позже появились идеи реализации систем управления темпоральными базами данных, до сих пор не воплощенные полностью ни в одной реализации крупных коммерческих СУБД[4]. Принято считать, что впервые идеи управления темпоральными данными появились в работе Якова Бен-Зви (Jacob Ben-Zvi) в 1982 г. [BZ82], но эта работа не получила широкой известности, хотя и предвосхитила многие последующие исследования в области темпоральных баз данных. Позже, в 80-е гг. начали появляться работы по темпоральной логике и использованию данных, зависимых от времени, их представлению внутри системы и визуализации для пользователя. С тех пор предлагались различные модели, создавались прототипы систем темпоральных баз данных ([Böh95]).

Одним из ключевых периодов в области исследований темпоральных баз данных, временем ее «официального» представления можно считать 1992–1993 гг. В это время сначала Ричард Снодграс (Richard Snodgrass) высказал идею о возможном темпоральном расширении стандарта языка запросов к реляционным базам данных SQL-92, а затем был проведен семинар «ARPA/NSF International Workshop on an Infrastructure for Temporal Databases», продемонстрировавший заинтересованность научного сообщества в создании и использовании темпорального расширения стандарта языка запросов к базе данных ([AJS95]). После этого семинара была образована комиссия по созданию спецификации подобного языка, в которую вошли Ричард Снодграс, Илсу Ан (Ilsoo Ahn), Гэд Ариав (Gad Ariav), Дон Бэтори (Don Batory), Джеймс Клиффорд (James Clifford), Картис Дайрсон (Curtis E. Dyreson), Кристиан Йенсен (Christian S. Jensen), Рамес Элмасри (Ramez Elmasri), Фабио Гранди (Fabio Grandi), Вольфганг Кафер (Wolfgang Kaefer), Ник Клайн (Nick Kline), Кришна Кулкарни (Krishna Kulkarni), Тинг Клифф Леунг (Ting Y. Cliff Leung), Никос Лоренцос (Nikos Lorentzos), Джон Роддик (John F. Roddick), Эрай Серев (Arie Segev), Майкл Су (Michael D. Soo) и Суринараяна Срипада (Surynarayana

[4] Дальше будут более подробно рассмотрены существующие реализации, но здесь подчеркивается, что они не решают все те проблемы, например, построение запросов, представление их результатов и оптимизацию хранения, которые обыкновенно рассматривают исследователи темпоральных баз данных.

M. Sripada). После нескольких лет плодотворной работы в начале 1994 года появилась первая предварительная версия спецификации языка, а на основе полученных замечаний в сентябре того же года была выпущена окончательная спецификация языка запросов TSQL2 [SAAB95].

Одновременно со спецификацией языка запросов TSQL2 были распространены комментарии [SAAB95], в которых объяснялись решения, принятые в ходе подготовки языка запросов, а также предоставлялись примеры, и рассматривались возможные пути реализации данного языка на основе существующих СУБД. Позднее эти комментарии, которые, по сути, не являлись частью спецификации языка, стали использоваться для сравнения TSQL2 с другими предлагаемыми темпоральными моделями и языками запросов к ним.

2.1. Принятие стандарта и коммерческие реализации

Представленная спецификация языка TSQL2 позволяла делать оптимистичные прогнозы относительно возможности расширения спецификации языка SQL-92 новыми конструкциями и одновременной интеграции темпоральной модели в реляционную модель данных. Разные члены сообщества исследователей темпоральных баз данных работали над переносом конструкций и логики языка TSQL2 в язык SQL3 (позднее названный SQL:1999). Первым шагом стало добавление в 1995 году проекта седьмой части спецификации языка SQL3, названной SQL/Temporal. В 1996 году два предложения [SBJ96a] и [SBJ96b] (а также их дополнения [Sno96] и [Sno97]), описывающие добавление действительного, или модельного (valid) и транзакционного (transaction)[5] времен к стандарту SQL-92, были единодушно поддержаны в ANSI и переданы в ISO. В это же время Андреас Стейнер (Andreas Steiner) и Михаель Бехлен (Michael H. Böhlen) создали прототип TimeDB, поддерживающий возможность работать и с действительным, и с транзакционным временами ([Tim07]). Система являлась прослойкой[6] между интерфейсом пользователя и одной из коммерческих СУБД, обеспечивая поддержку темпоральных конструкций путем их преобразования в обычные реляционные структуры и обратно. Позднее разработчики пытались позиционировать TimeDB как коммерческий проект, но, по-видимому, не достигли в этом успехов.

Многие исследователи были убеждены, что окончательное принятие стандарта и появление расширений для поддержки темпоральных баз данных в развитых коммерческих СУБД должны были произойти уже в самом начале XXI века. Однако седьмая часть стандарта так и осталась на бумаге. Более того, несколько лет назад работы над стандартом SQL/Temporal были вообще

[5] Эти и другие термины будут более подробно рассмотрены и разъяснены в разд. 4.

[6] Подробнее об этом см. разд. 7.

прекращены (или приостановлены на неопределенный срок) [Jcc07]. Производители СУБД также не добавили в свои продукты полноценную темпоральную поддержку. Кристофер Дейт (C. J. Date) в [DDL02] считает целесообразным введение курса по работе с темпоральными данными в рамках программы подготовки специалистов (и сам читает подобный курс в одном из университетов), но соглашается, что для производителей сейчас более актуальными являются другие задачи, связанные, например, с расширением функциональности для поддержки электронного бизнеса и т.п. Однако стоит отметить, что некоторые производители коммерческих СУБД уже обращают внимание на проблему поддержки работы с темпоральными данными в своих системах, предоставляя определенные дополнительные возможности как для администраторов, так и для пользователей. Но из-за отсутствия стандарта нет ни единого подхода, ни одинаковой поддерживаемой функциональности. Более того, системы в явном виде не позиционируются как поддерживающие работу с темпоральными данными, поэтому невозможно говорить даже о наличии стандарта de facto.

Между тем, начиная уже с 2000 года, часть исследователей темпоральных баз данных обратила внимание на XML-данные, для которых стандарт языка запросов еще только вырабатывался. Необходимость разработки новых интерпретаторов запросов и гибкость структур языка давали некоторым разработчикам надежду на реализацию темпоральной XML-СУБД. Однако текущая ситуация напоминает ту, которая сложилась в области реляционных баз данных: предложено несколько моделей и созданы прототипы, но все это осталось пока лишь иллюстрацией или инструментом для решения конкретной проблемы и не может претендовать на то, что со временем будет стандартизовано[7].

3. Появление области исследований темпоральных баз данных

В предыдущем разделе была представлена краткая история появления и развития области исследований темпоральных баз данных. Далее будут даны определения основных понятий, описаны наиболее важные проблемы, с которыми сталкивались и сталкиваются разработчики при создании темпоральных баз данных. Но вначале кратко ответим на один из основных вопросов, почему же возникла потребность в расширении стандарта языка SQL, и почему возникла подобная область исследований?

Одной из предпосылок создания темпорального расширения языка SQL явилось то, что почти все приложения, использующие реляционные базы данных, являются темпоральными по своей сути. В базах данных хранятся данные, изменяющиеся с течением времени. С другой стороны, многие

[7] Здесь речь идет об общем стандарте, а не какой-нибудь специальной функциональности для определенной прикладной области.

понимают, что в языке запросов SQL отсутствует адекватная и эффективная поддержка работы с темпоральными базами данных. Традиционная реляционная база данных хранит информацию лишь о текущем состоянии, и СУБД не предоставляет возможности работать с данными, привязанными к определенным датам или интервалам времени (то есть темпоральными данными). Поэтому почти во всех базах данных поддержка работы с такими данными обеспечивается усилиями программистов и администраторов, которые используют для решения задач «неудобные» конструкции языка. При этом многие простые запросы, которые легко формулируются «вне времени», довольно тяжело переписать для «самодельной» темпоральной системы, и они получаются достаточно громоздкими, что чревато появлением ошибок.

4. Основные понятия

В данном разделе будут представлены основные понятия, используемые в области темпоральных баз данных. Начнем с базового понятия – *линия времени*.

4.1. Линии времени

В повседневной жизни человек чаще всего не задумывается, что использует только одну линию времени. События могли уже произойти в прошлом или только планируются в будущем, но время всегда измеряется согласно одним часам. С другой стороны, в базе данных может сохраняться информация о событиях и интервалах времени, соответствующих различным представлениям и связям. Если обработкой подобных данных занимается сам пользователь, то используемый тип времени можно назвать временем, определяемым пользователем. Его отличительным признаком служит отсутствие интерпретации со стороны СУБД, так как обработка данных, связанных со временем, полностью возлагается на пользователя. Фактически, все современные СУБД обеспечивают поддержку подобной разновидности времени, например, с помощью введения специальных типов данных DATA или TIMESTAMP.

Если рассматривать данные, представленные в базе данных, как некоторое отражение текущего состояния действительности для моделируемого мира, то каждая запись может восприниматься как некоторый факт, который является истинным в определенный момент или интервал времени. При переходе к темпоральной базе данных для каждого факта можно указать тот промежуток времени[8], когда этот факт являлся истинным в моделируемом мире, представленном в базе данных. Подобное представление времени, когда с

[8] Вместо того чтобы говорить об интервалах истинности факта, можно говорить о фактах, истинных в определенный момент времени. Это два различных взгляда на одну и ту же ситуацию: *интервальный* и *точечный*. Более подробно они будут рассмотрены ниже.

данными связывается промежуток времени их актуальности (с точки зрения моделируемого мира), называется *модельным*, или *действительным* (*valid*) временем. Его значения можно сравнить с показаниями часов моделируемого мира. Поскольку довольно часто в базе данных отражается именно реальный мир, могут быть заданы соотношения между значениями времени реального мира и представленной в базе данных моделью. Отметим, что значениями данного типа времени могут быть моменты времени как в прошлом, так и в будущем. Кроме того, эти значения могут изменяться, то есть истинность факта в определенные моменты времени может приниматься или отклоняться.

Другим типом линии времени, который рассматривается исследователями темпоральных баз данных, является *транзакционное* время. В любой СУБД каждой записи базы данных можно сопоставить тот промежуток времени, когда данная запись была представлена в базе данных, т.е. промежуток времени между моментами добавления записи и ее удаления из базы данных. При этом отметим, что операция обновления, которая действительно вносит изменения в запись, понимается как составная операция удаления старой записи и добавления новой. Очевидно, что значения транзакционного времени не могут относиться к будущему. В подавляющем числе СУБД транзакционное время используется для работы с блокировками, журналом для восстановления системы. В некоторых системах администраторы даже могут использовать специальные расширения языка SQL, позволяющие получить доступ к транзакционному времени и истории изменений записей в базе данных.

сотрудник	з/плата
ALAN	500
BOB	400

Рис. 1. Таблица без темпоральных расширений

сотрудник	з/плата	действительное время
ALAN	500	с 1 января 2006
BOB	300	с 1 марта 2005 по 31 января 2006
BOB	400	с 1 февраля 2006

Рис. 2. Таблица с поддержкой действительного времени

Чтобы ответить на вопрос, как соотносятся между собою модельное и транзакционное время, рассмотрим следующий пример. Пусть имеется таблица, в которой хранится информация о текущей зарплате сотрудника

(рис. 1). При наличии поддержки действительного времени мы могли бы в любой момент сказать, какая у сотрудника была зарплата за произвольный период времени (рис. 2). Таким образом, данные о зарплате могут быть представлены как последовательность изменяющихся значений. При наличии поддержки транзакционного времени мы могли бы сказать, в какой момент в таблицу были внесены изменения[9] (рис. 3).

табельный номер	з/плата	транзакционное время
ALAN	500	с 20 декабря 2005
BOB	300	с 3 марта 2005 по 27 января 2006
BOB	500	с 28 января 2006 по 5 февраля 2006
BOB	400	с 6 февраля 2006

Рис. 3. Таблица с поддержкой транзакционного времени

табельный номер	з/плата	действительное время	транзакционное время
ALAN	500	с 1 января 2006	с 20 декабря 2005
BOB	300	с 1 марта 2005 по 31 января 2006	с 3 марта 2005 по 27 января 2006
BOB	500	с 1 февраля 2006	с 28 января 2006 по 5 февраля 2006
BOB	400	с 1 февраля 2006	с 6 февраля 2006

Рис. 4. Таблица с поддержкой обеих линий времени

Теперь предположим, что для таблицы поддерживается как действительное, так и транзакционное время (рис. 4). Тогда в случае, если неправильно введенные данные были впоследствии исправлены, можно будет точно сказать, когда это было сделано. Кроме того, информация о подобных изменениях необходима, так как некорректные данные могли бы быть уже

[9] Можно отметить, что уже на этом примере заметна разница между действительным и транзакционным временем, так как у сотрудника повышается заработная плата, например, с первого числа месяца, а не со второго, когда данные были реально внесены в систему. При наличии поддержки только транзакционного времени можно было бы сказать, когда были изменены актуальные данные в системе, но нельзя было бы сказать, когда эти данные необходимо было бы уже учитывать при вычислениях.

использованы в каких-нибудь отчетах. Поэтому в данном случае требуется поддержка транзакционного времени. При обновлении значений в системе (даже в случае исправления ошибки в данных) интервал транзакционного времени также обновляется, поэтому можно просмотреть список изменений в базе данных.

Следовательно, временные метки транзакционного времени предоставляют информацию о времени изменения данных или исправления ошибок, а временные метки действительного времени хранят информацию об изменении некоторых параметров моделируемого мира. Таким образом, модельное и транзакционное время оказываются ортогональными друг другу (рис. 5).

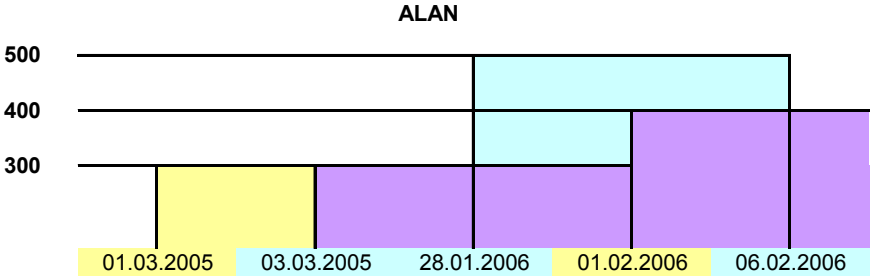

Рис. 5. Связь линий времени для сотрудника ALAN

Исследователи темпоральных баз данных обычно используют один из данных типов времени или оба одновременно. В некоторых работах предлагаются и другие линии времени[10], хранение значений которых может быть интересно пользователю, но все они могут быть сведены к одному из рассмотренных типов, возможно, через дополнительные отношения.

Говоря о линиях времени, необходимо ввести еще один термин – *гранулярность*, которая показывает, насколько близкие моменты на оси времени все еще будут отличимыми друг от друга. Например, возможно, что для данных о заработной плате сотрудника достаточно разбиения по дням, но для транзакционного времени может быть недостаточно даже разбиения по секундам, если в СУБД возможна более частая фиксация транзакций.

В общем случае с каждой линией времени может быть еще связан некоторый *календарь*, который определяет диапазоны значений, гранулярность,

[10] В качестве примера приведем «время переноса значений из другой системы». Если предположить, что у нас есть поддержка транзакционного времени, то при необходимости копировать содержимое из одной базы данных в другую оказывается, что факт уже известен в одной системе, но не известен в другой, поэтому предлагается использовать некоторое подобие второго транзакционного времени.

соответствия и преобразования между моментами времени для различных осей времени.

4.2. Интервальное и точечное представления

В предыдущем подразделе при обсуждении действительного времени, говорилось, что существует некоторый интервал, в котором определенный факт являлся истинным. Это так называемое *интервальное* представление. Однако можно рассматривать отдельный момент времени и все факты, которые были истинны в этот конкретный момент (рис. 6).

	действительное время		транзакционное время	
	ALAN	**BOB**	**ALAN**	**BOB**
19.12.2005				
20.12.2005			100	
…			100	
31.12.2005			100	
01.01.2006	100		100	
…	100		100	
28.02.2005	100		100	
01.03.2005	100	300	100	
02.03.2005	100	300	100	
03.03.2005	100	300	100	300
…	100	300	100	300
15.09.2005	**100**	**300**	**100**	**300**
…	100	300	100	300
27.01.2006	100	300	100	300
28.01.2006	100	300	100	500
…	100	300	100	500
31.01.2006	100	300	100	500
01.02.2006	100	400	100	500
…	100	400	100	500
05.02.2006	100	400	100	500
06.02.2006	100	400	100	400
…	100	400	100	400
сейчас	100	400	100	400

Рис. 6. Точечное представление и срезы на линии времени

Здесь говорится о представлении времени с точки зрения пользователя, то есть тех условных моделях, в рамках которых могут формулироваться запросы и возвращаться их результаты. При использовании любого из этих представлений истинность фактов не меняется, но в случае *точечного* представления мы получаем срез всех фактов на какой-то конкретный момент времени, а для интервального представления нас интересует определенный факт и периоды его истинности. Если говорить об обычной реляционной модели, то она опирается на точечное представление для актуального

состояния данных. В ряде работ проводится сравнение этих подходов [Tom96], исследуются возможности их совместного использования и объединения [BBJ98], [TS01], а также анализируются способы эффективной реализации менее распространенных точечных подходов [Tom98].

5. Расширение реляционной модели и СУБД

Реляционная модель данных оказалась очень подходящей для хранения данных, обработки и представления результатов запросов, и поэтому темпоральные базы данных с самого начала предполагалось основывать на существующей реляционной модели[11], так как темпоральное расширение является лишь одним из дополнительных признаков хранимых данных.

5.1. Создание темпоральной СУБД

Исследователи темпоральных баз данных рассматривали несколько способов создания темпоральной СУБД. Вначале рассматривалась возможность создания темпоральных СУБД «с нуля», то есть самостоятельная реализация некоторой темпоральной модели. Однако реляционные СУБД быстро прогрессировали, расширялась их функциональность, поэтому создание «урезанной» темпоральной СУБД было нецелесообразным, а ресурсов и возможности повторной реализации всех средств реляционной СУБД (при создании темпоральной СУБД) просто не было.

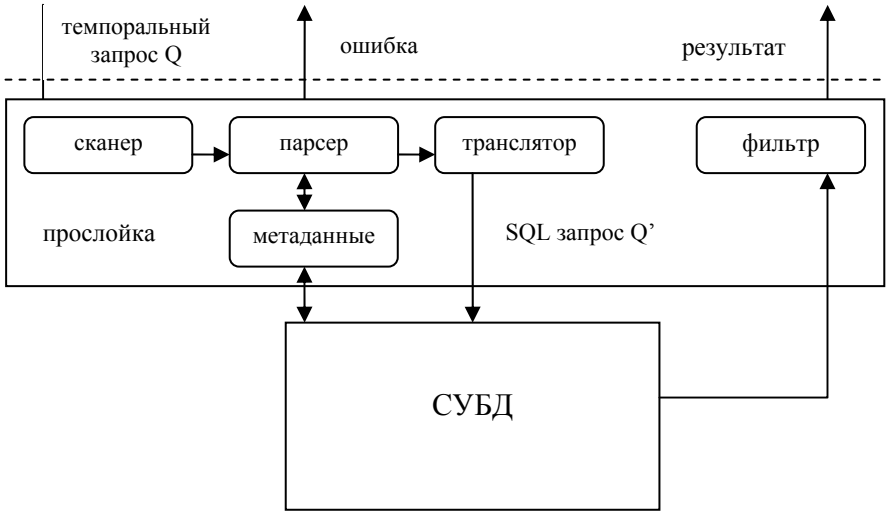

Рис. 7. Общий вид многоуровневой архитектуры темпоральной СУБД

[11] В данной статье основное внимание уделяется именно реляционным темпоральным базам данных. О других вариациях будет упомянуто в разд. 6.

Отсутствовала и соответствующая потребность со стороны пользователей. Поэтому вскоре разработчики стали рассматривать различные способы дополнения и расширения обычных реляционных СУБД поддержкой темпоральной модели данных[12]. Почти все такие способы сводились к созданию некоторого функционального блока, отвечающего за разбор темпоральных запросов, подмену их некоторыми реляционными вычислениями, а потом обратное преобразование в требуемое темпоральное представление для возвращения результатов пользователю. Основным отличием был уровень «вмешательства» в реляционную СУБД, а также степень интеллектуальности данного блока темпоральных преобразований (рис. 7). Сравнение различных вариантов реализации промежуточного слоя представлено, например, в [TJS98].

Рассмотрим три крайних случая: преобразование на уровне ядра реляционной СУБД, преобразование на уровне пользовательского приложения и использование независимого промежуточного proxy-уровня. Первый способ предоставляет максимум возможностей по расширению синтаксиса языка баз данных, обеспечению различных проверок, а также оптимизации. Он также обеспечивает полную прозрачность для всех пользовательских приложений, а возможно, и для администратора БД. Однако он доступен только разработчикам СУБД.

В отличие от простого использования приложением реляционной СУБД для работы с темпоральными данными второй способ предполагает выделение некоторых модулей или библиотеки (в пользовательском приложении), отвечающих за преобразование именно темпоральных запросов. То есть основное приложение использует некоторую темпоральную абстракцию, а не реляционную БД в чистом виде, а потом интерпретирует результаты запросов. Подобная абстракция (пусть и на уровне приложения) позволяет сократить число ошибок и отделить бизнес-логику приложения от технической составляющей хранения данных[13]. В [Sno99] подробно описано, как можно создавать темпоральные приложения в рамках реляционной СУБД.

Третий способ подразумевает создание между пользовательским приложением и СУБД некоторого промежуточного «черного ящика» (будем называть его proxy-уровнем), который может быть реализован в виде драйвера, сервиса, внешней библиотеки и т.п. Для пользовательского приложения этот proxy-уровень должен работать как темпоральная СУБД. С другой стороны, для СУБД этот proxy-уровень является обычным реляционным приложением. Поэтому proxy-уровень оказывается прозрачным как для пользовательского приложения, так и для СУБД, а также не требует

[12] Фактически, именно этот факт приводит к «реляционности» большинства темпоральных моделей.

[13] Возможна ситуация, когда информация, относящаяся к различным периодам времени, будет храниться на различных носителях.

внесения изменений в их исходный код. Исходя из того, что между proxy-уровнем и СУБД интенсивность обмена данными может оказаться на порядок выше, чем между proxy-уровнем и приложением (с учетом различных дополнительных оптимизаций), предпочтительно размещать его как можно ближе к СУБД.

Рассмотрим преимущества и недостатки представленных способов. Основным недостатком первых двух подходов является необходимость в изменении кода СУБД или приложения, и поэтому они доступны, в первую очередь, самим разработчикам. Одним из недостатков второго и третьего способа является невозможность сильно влиять на внутреннее представление и хранение информации в СУБД, оптимизацию доступа к ней и т.п. Одним из значительных преимуществ всех трех способов является возможность использовать уже существующие в СУБД модули интерпретации и обработки, лишь дополняя их разбором и преобразованием только темпоральных конструкций и элементов. Дополнительным преимуществом третьего способа можно назвать относительную независимость от конкретной СУБД, если не используются какие-либо специфические особенности и конструкции. В большинстве случаев на работоспособность proxy-уровня не влияет обновление версии СУБД, так как синтаксис SQL-запросов останется прежним.

5.2. Язык запросов к темпоральной базе данных

Для работы с темпоральными базами данных необходимо было разработать язык запросов к ним, который должен был бы стать расширением языка запросов SQL к реляционным базам данных. Рассмотрение языка запросов начнем с конструкций выборки, предложенных авторами языка TSQL2 (SQL/Temporal).

Выше говорилось о темпоральных базах данных, однако реляционная модель предполагает, что данные хранятся в таблицах, и база данных состоит из набора таких таблиц. Поэтому уточним понятия. Во-первых, поскольку практически любая база данных содержит данные, связанные с промежутками времени, ее можно было бы назвать темпоральной. Однако, говоря о темпоральной СУБД, подразумевают, что интерпретацией подобных данных занимается сама система, и поэтому принято считать, что темпоральная база данных – это база данных, содержащая связанные со временем данные, интерпретацией которых занимается СУБД, являющаяся, таким образом, темпоральной СУБД[14]. С другой стороны, под управлением темпоральной СУБД вполне может находиться обычная реляционная база данных. Более того, для части таблиц темпоральная поддержка может быть включена, а для

[14] Данное определение не противоречит определениям, приведенным ранее, но вполне конкретизирует пару «темпоральная база данных и соответствующая СУБД».

других таблиц – нет[15]. Поэтому далее будем рассматривать отдельную таблицу, на время забывая о возможных ее связях с другими таблицами.

При построении модели языка запросов к темпоральной базе данных ставились следующие цели. Во-первых, при переходе к темпоральной модели желательно расширить все три компонента модели данных: структуру данных, операции и ограничения целостности. Во-вторых, любая корректная конструкция в исходном языке запросов должна остаться корректной в новом языке, и семантика этих конструкций должна остаться прежней; например, результат, возвращаемый запросом, должен быть таким же. То есть необходимо обеспечить восходящую совместимость языка запросов и базы данных. Кроме того, желательно обеспечить темпоральную восходящую совместимость: необходимо, чтобы после добавления в систему темпоральной поддержки все унаследованные конструкции продолжали работать так же, как и раньше. Из этого следует, что все существующие приложения не должны почувствовать переход от обычной реляционной СУБД к темпоральной СУБД ([BBJ+97]). Более того, последующее добавление темпоральной поддержки в какую-нибудь конкретную таблицу не должно отразиться на корректности выполнения запросов. С другой стороны, подобное добавление должно позволить использовать темпоральные запросы в новых программах, заметно облегчая работу программистам и администраторам системы.

Рассмотрим таблицу, содержащую информацию о заработной плате сотрудников. До добавления темпоральной поддержки таблица содержала только актуальную информацию (на текущий момент времени)[16]. В таком случае у приложения имеется возможность узнать текущее значение любых элементов данных, но предыдущее значение будет потеряно после любого изменения. После удаления записи например вообще невозможно будет сказать, , сколько получал данный сотрудник перед увольнением. Такая ситуация существует при использовании традиционной реляционной модели (рис. 8). Предположим, что позже было принято решение о необходимости хранить историю значений заработной платы сотрудников. Решить это можно было бы несколькими путями. Во-первых, можно создать специальную таблицу с историей зарплаты, но это могло бы быть достаточно странным, так как уже есть таблица «заработных плат». Более того, пришлось бы модифицировать код приложения, чтобы при удалении и изменении значений происходило автоматическое обновление данных и во второй таблице.

[15] Это имеет решающее значение при проектировании базы данных, а также при построении запросов с использованием соединений или программировании вложенных циклов.

[16] Можно возразить, что в системе с самого начала должна была бы храниться вся история изменений заработной платы сотрудника, а таблица только с актуальными показателями малопригодна. Но это лишь подчеркивает, что при разработке приходится учитывать неявную темпоральную составляющую.

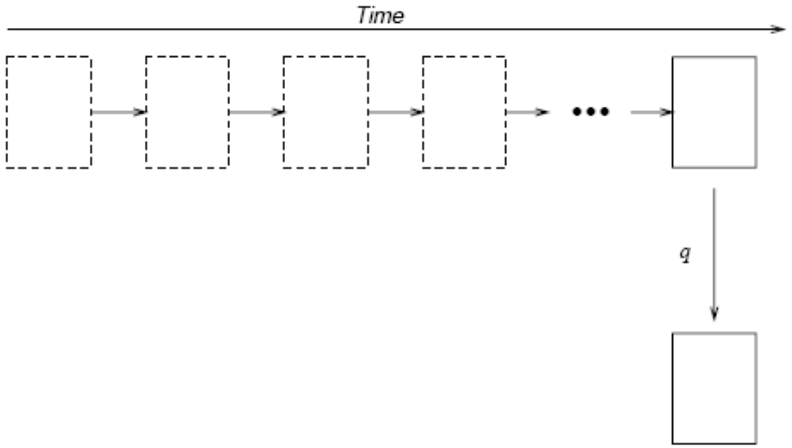

Рис. 8. Работа с таблицей без темпоральной поддержки

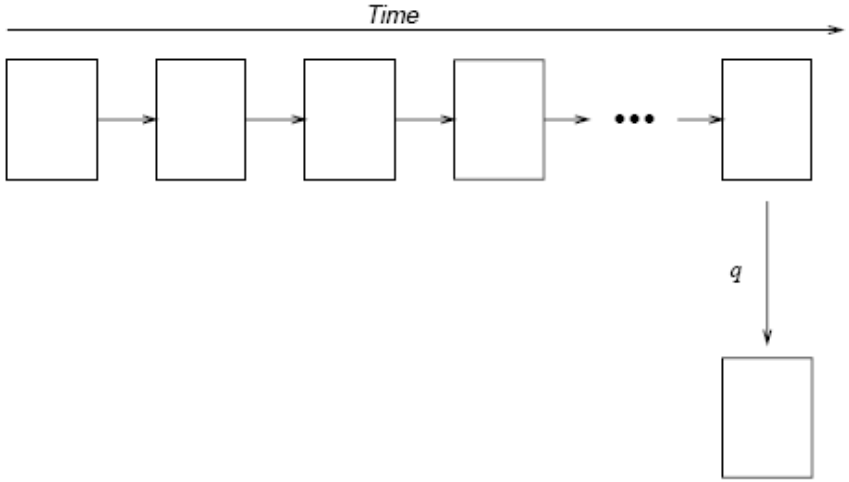

Рис. 9. Работа реляционных приложений с темпоральной таблицей

Второй способ – добавление специальных столбцов, отражающих интервал актуальности конкретной записи, но при этом также потребуется доработка приложения, и логика многих запросов может стать не слишком очевидной. Наиболее естественным было бы добавить для данной конкретной таблицы темпоральную поддержку. В этом случае все ранее разработанные запросы будут корректно работать (с текущим состоянием), но можно сформулировать

новые темпоральные запросы, которые позволят узнать историю изменений (рис. 9).

Какие же запросы могут быть сформулированы к таблице с темпоральной поддержкой? Во-первых, это запрос значений на какой-нибудь момент времени в прошлом, то есть создание среза истинности фактов на произвольную дату. Например, для обычного реляционного запроса «какую зарплату сейчас получает каждый из сотрудников?» можно легко сформулировать его темпорального двойника «какую зарплату получал каждый из сотрудников в указанную дату?» В этом случае результат запроса останется в рамках реляционного представления. С другой стороны, вполне естественным оказывается запрос «когда и какую зарплату получал каждый из сотрудников?». Здесь уже в результатах запроса появляется линия времени. Один из традиционных способов представления подобных результатов опирается на интервальное действительное время, то есть к обычному реляционному представлению результатов добавляется еще один столбец, содержащий интервалы истинности фактов. Алгоритм формирования результатов подобных запросов можно упрощенно представить следующим образом: для каждого момента времени[17] вычисляется реляционный подзапрос «какую зарплату получает каждый из сотрудников», после чего к общему результату добавляются результаты этих подзапросов с учетом интервалов истинности. Подобная семантика «последовательной» интерпретации реляционных запросов называется *последовательной* (*sequenced valid semantics*), см. рис. 10.

Однако на основе этой семантики невозможно сформировать запрос, который требует «сравнения» нескольких последовательных моментов времени. К таким запросам относится большинство запросов, включающих агрегационные функции «во времени», например, «вывести среднюю заработную плату сотрудника за все периоды времени» (отметим, что результат будет представлен обычной реляционной таблицей) или «вывести периоды, когда на оплату труда сотрудников уходила максимальная сумма». Предусмотреть все подобные типы запросов невозможно, и также нельзя ограничивать выразительные возможности языка, поэтому была предложена конструкция запросов *произвольного* доступа к темпоральным данным (*Non-Sequenced Queries*), которые предоставляют возможность самостоятельно сформулировать необходимый запрос, накладывая ограничения на системный темпоральный столбец[18]. См. рис. 11.

[17] Достаточно рассматривать только те моменты времени, когда происходит изменение истинности одного из фактов, хранящихся в базе данных.

[18] Фактически, система позволяет только формулировать ограничения и отношения между интервалами и отдельными моментами времени, а все остальное отдается на откуп пользователю.

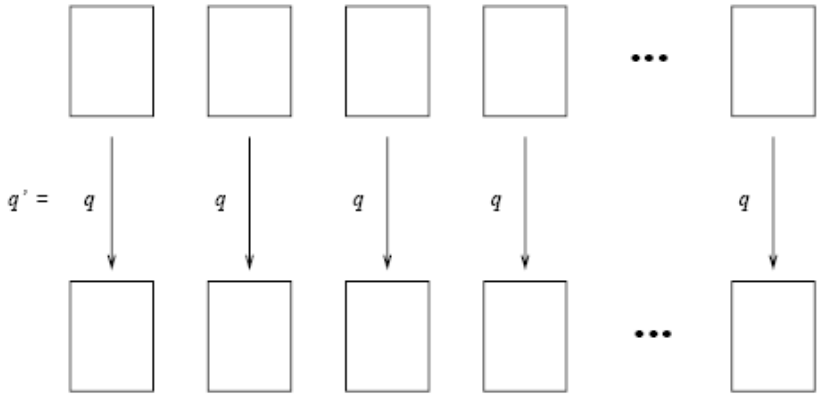

Рис. 10. Обработка "последовательных" запросов к темпоральной таблице

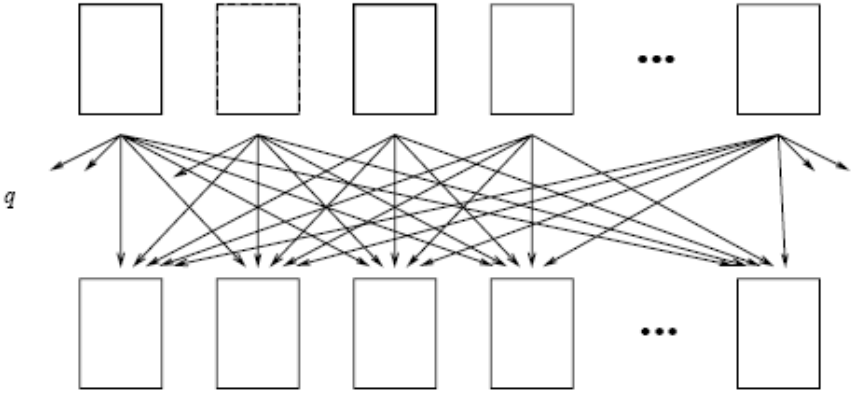

Рис. 11. Обработка "произвольных" запросов к темпоральной таблице

Таким образом, при наличии темпоральной поддержки в системе можно выделить четыре уровня использования запросов (для конкретной таблицы): 1 – темпоральная поддержка для таблицы отсутствует, 2 – использование реляционных запросов к таблице с темпоральной поддержкой, 3 – использование последовательных запросов и 4 – использование произвольных темпоральных запросов. Подобная классификация верна как для таблиц с поддержкой действительного времени, так и для тех, для которых поддерживается транзакционное время. Поскольку эти линии времени являются ортогональными и независимыми, в системе с темпоральной

поддержкой получается шестнадцать вариантов использования запросов на выборку.

На основе запросов на выборку можно формулировать темпоральные ограничения целостности, которые будут проверяться при операциях обновления базы данных. Для операций модификации и удаления записей можно накладывать аналогичные ограничения в их разделах WHERE. С другой стороны, обработка запросов на обновление для транзакционного и действительного времени будет принципиально отличаться. Например, в разделе SET невозможно задавать системные поля транзакционного времени. Поэтому полностью корректная реализация поддержки транзакционного времени возможна только при реализации темпоральных операций внутри ядра СУБД.

5.3. Представление результатов темпоральных запросов

Выше уже упоминались два подхода к представлению информации в темпоральной базе данных: интервальный, когда принимается во внимание период истинности факта, и точечный, когда для каждого отдельного момента времени рассматриваются все факты, истинные в этот момент. При реализации темпоральной СУБД и проектировании языка запросов требуется выбрать один из этих вариантов представлений. При хранении данных в базе данных наиболее компактным является интервальное представление. При представлении результатов запросов также многое говорит в пользу интервального представления[19]. Но при формулировке многих запросов удобнее сравнивать моменты времени, а не оперировать интервалами истинности отдельных фактов, поэтому точечное представление получает значительное преимущество.

Операции над множествами очень удобны для теоретических выкладок и формализации, но трудно реализуемы в системе при значительном объеме данных. Поэтому интервалы времени принято разделять на такие подинтервалы, чтобы для каждого факта они не пересекались, но дополняли друг друга до исходного интервала, а любые два интервала разных фактов либо не пересекались, либо совпадали. При таком представлении истинность фактов не будет меняться ни для одного из получившихся интервалов, что позволит, например, корректно использовать полный вложенный перебор по интервалам и фактам при получении результата запросов. После получения результата в виде набора интервалов истинности для каждого факта необходимо выполнить обратную процедуру – объединение пересекающихся или примыкающих друг к другу интервалов. Два данных преобразования называются *распаковкой* и *упаковкой*. Эти операции могут применяться и во

[19] Точечное представление может быть более удобно при ограниченности (и дискретности) возвращаемых моментов времени и их последующем использовании при программной обработке.

время промежуточных вычислений. Фактически, перед каждой операцией над темпоральными данными можно выполнять распаковку, а потом – упаковку.

5.4. Специальное значение «сейчас»

Слово «сейчас» означает «в настоящее время, в данный момент». В запросах на языке SQL довольно часто используются конструкции CURRENT_TIMESTAMP, CURRENT_DATE и CURRENT_TIME, которые заменяются соответствующими константами в момент выполнения запросов, поэтому в самой базе данных хранятся лишь конкретные значения дат и времени. Так как язык запросов к темпоральной базе данных является расширением языка SQL, данные конструкции в нем также могут использоваться как синонимы «динамических» констант. Однако для работы с фактами, которые истинны в настоящий момент, но могут меняться с течением времени, необходимо использовать дополнительное специальное значение *СЕЙЧАС*. Оно используется при задании верхней границы интервала истинности факта, хранимого в таблице с темпоральной поддержкой ([CDI+97]).

Пусть в базе данных хранится информация о периоде работы сотрудника в компании. Тогда до 2 апреля для сотрудника, пришедшего на работу 1 апреля, промежутком времени его работы в компании является один день 1 апреля. На следующий день промежуток его работы в компании изменится на интервал с 1 по 2 апреля. Через день интервал примет вид 1 – 3 апреля и так далее. Одним из путей решения подобной проблемы могло бы быть ежедневное обновление базы данных, изменяющее верхнюю границу интервала на корректное значение, но это нереальный вариант. Поэтому при выполнении запросов можно использовать подобную динамическую подстановку[20]. Для этого можно хранить динамическую константу *СЕЙЧАС*, обозначающую зависимость факта от текущего времени. В приведенном примере это бы означало, что человек работает в компании с 1 апреля по *СЕЙЧАС*.

В качестве представления *СЕЙЧАС* можно использовать одно из значений домена TIMESTAMP или пустое значение NULL. При выборе значения для представления *СЕЙЧАС* необходимо гарантировать, чтобы это значение не использовалось с каким-либо иным смыслом. Фактически, требуется сделать выбор из значения NULL и максимального или минимального значения типа TIMESTAMP. При использовании *СЕЙЧАС* в запросах необходимо принимать во внимание, что его следует заменять текущим временем на этапе выполнения. При работе над системой TimeDB [TIM07] исследовалась производительность системы для различных вариантов представлений. Было продемонстрировано, что представление *СЕЙЧАС* с помощью минимального

[20] Если подстановка для CURRENT_TIMESTAMP, CURRENT_DATE и CURRENT_TIME выполняется для данных запроса, то здесь подстановка выполняется для информации, хранимой в базе данных.

значения типа TIMESTAMP всегда самое медленное. Представление в виде NULL было наиболее эффективным при большем проценте пересечений с текущим временем. Однако представление в виде NULL-значения всегда приводило к максимальному количеству чтений диска. На основе этого анализа был сделан выбор в пользу максимального значения типа TIMESTAMP.

Использование некоторого специального значения *СЕЙЧАС* не является единственно возможным решением. Например, можно разделить все факты на два класса, в одном из которых хранятся факты, для которых интервал уже точно известен, а в другом – те, у которых верхняя граница еще не известна. Если хранить эти факты в разных таблицах, то не потребуется вводить специальное значение *СЕЙЧАС*, однако несколько усложнится выборка, а таблицы с поддержкой и транзакционного, и действительного времени придется разбивать на четыре подтаблицы.

Выше речь шла о представлении значения *СЕЙЧАС* в системе, однако важна и его интерпретация. Например, тот факт, что сотрудник работает с 1 апреля по *СЕЙЧАС*, не подтверждает, но и не опровергает, что на следующей неделе он также будет работать. Но для другого сотрудника может быть известно, что он работает только до конца этой недели, и поэтому в отчетах за данную неделю о сотрудниках, которые будут работать на следующей неделе, второй сотрудник фигурировать не должен, а про первого сказать ничего нельзя. Поэтому корректная интерпретация значения *СЕЙЧАС* возможна лишь для прошлого. При работе с событиями будущего необходимо проявлять осторожность и точно понимать, что и каким образом должно быть получено в качестве результата задаваемого запроса.

5.5. Разрежение таблиц с темпоральной поддержкой

Если говорить о таблицах с темпоральной поддержкой, то причиной снижения производительности может стать постоянно возрастающий объем хранимых данных. Это, в первую очередь, относится к таблицам с темпоральной поддержкой транзакционного времени, так как в таких таблицах данные физически не удаляются из системы, а лишь добавляются (в том числе и при модификации записей). С другой стороны, через определенный промежуток времени оказывается, что часть данных устарела, и поэтому их было бы желательно физически удалить из системы, так как они не только занимают место, но и снижают производительность при выборках. Но подобное физическое удаление подвергает риску общую целостность хранящихся данных.

Поэтому фундаментальным требованием является возможность контролирования физического удаления данных, что приводит к операции, которая называется *разрежением* (*vacuuming*) таблиц с темпоральной поддержкой. Для обеспечения корректности выполнения разрежения необходимо иметь возможность указывать в запросе данные, которые должны

быть удалены, и те, которые должны быть оставлены. Очевидно, что все «активные» строки должны остаться в таблице при любом ее разрежении. В простейшем случае отбрасываются все те кортежи, которые были удалены[21] до определенного момента времени – происходит срез истории.

Но подобное разрежение является не совсем корректной операцией. Например, пусть мы хотим сделать выборку состояния на какой-нибудь момент времени *A*, предшествующий аргументу среза *B* (*A* < *B*), когда в таблице находились некоторые строки *X* и *Y*, первая из которых была удалена до момента времени *B*, а вторая – нет. Тогда в качестве результата запроса будет выдана строка *Y*, но не *X*. Многие запросы могут начать неправильно работать, и к ним, скорее всего, будут относиться все запросы, использующие агрегатные функции.

Чтобы обезопасить себя от части подобных проблем, можно предложить специальный вариант среза, когда нижняя граница интервала транзакционного времени для всех строк становится равной моменту времени среза, если этот момент принадлежал исходному интервалу. Но и в этом случае искажается информация о времени истинного появления строки в системе. Поэтому разрежение темпоральных таблиц не является однозначным решением, и в каждом конкретном случае необходимо отдельно рассматривать все плюсы и минусы.

Отметим, что с точки зрения работы с системой и формулировки запросов нет необходимости в разрежении, однако переполнение базы данных «устаревшими» сведениями может негативно сказываться на производительности. От негативных последствий проблемы зависимости поведения приложений при проведении разрежения можно частично избавиться, если вместо реального удаления данных выполнить их перенос на другой доступный носитель. Например, данные из одной базы переносятся в другую, расположенную на другом сервере, причем в исходной базе остается информация об этом переносе. В этом случае при необходимости обращения к истории запрос может быть перенаправлен или объединен с результатом выборки из другой базы данных.

Проблема с «разрежением» таблиц относится к транзакционному времени, но она также рассматривается и с точки зрения действительного времени. Например, если речь идет о хранении некоторых промежуточных версий какого-либо документа, то может оказаться, что интерес представляет лишь окончательная версия за январь прошлого года, а все промежуточные версии можно удалить. В этом случае можно также воспользоваться разрежением, но выбор доступных методов гораздо шире, например, можно оставить каждый третий документ или не больше одного документа в месяц, а само разрежение применить к документам прошлого года.

[21] В терминах транзакционного времени.

5.6. Проектирование темпоральных баз данных

В настоящее время существуют проверенные методы и инструментарий, используемые при проектировании реляционных баз данных. При работе с темпоральными базами данных необходимо учитывать темпоральную специфику и использовать соответствующие средства. Рассмотрим простой вопрос: сколько в базе данных должно быть таблиц с темпоральной поддержкой? Если для действительного времени ответ на данный вопрос достаточно прост (количество таблиц определяется потребностью приложений), то при наличии транзакционного времени он становится нетривиальным, так как от ответа от него зависит объем накапливаемой истории. При этом не всегда требуется именно поддержка транзакционного времени, а вполне может подойти и простой журнал, реализованный вне базы данных. При проектировании таблиц с поддержкой действительного времени можно использовать обычные инструменты, дополняя таблицы интервалами (парой значений) времени, однако в этом случае количество связей между таблицами заметно увеличивается. Более того, невозможно корректно описать работу со специальным значением *СЕЙЧАС*. При проектировании темпоральной базы данных следует также помнить, что все «обычные» реляционные ключи таблиц будут неявно расширяться верхней границей интервала транзакционного и/или действительного времени. Ограничения целостности также должны формулироваться с учетом этих обстоятельств ([JT95]).

Если темпоральная поддержка используется не для всех таблиц, то возникает вопрос, связанный с выборкой данных из нескольких таблиц на какой-либо момент прошлого: что делать с совместными выборками из таблицы с поддержкой транзакционного времени и из таблицы без подобной поддержки? Вполне возможно, что результат выборки будет неверен, так как информация из обычной таблицы могла быть уже удалена или изменена, а в соответствующей темпоральной таблице остались внешние связи. Таким образом, если разрешить подобные выборки, то могут возвращаться некорректные результаты. Если же такие выборки запретить, то становится непонятно, как работать с потенциально константными данными, например, названиями дней недели, которые могут храниться в таблице без темпоральной поддержки. Принудительное добавление темпоральной поддержки для всех таблиц также не является лучшим выходом, так как усложняет алгоритм работы СУБД и требует дополнительных ресурсов.

5.7. ACID-свойства темпоральных транзакций

Основными свойствами традиционных транзакций, поддерживаемыми в реляционных СУБД, являются ACID (атомарность, целостность, изолированность и продолжительность). При создании темпоральной СУБД чрезвычайно важно перенести подобные свойства и на темпоральные транзакции. ACID-свойства темпоральных SQL-транзакций могут быть

сохранены за счет отображения каждой темпоральной транзакции в одиночную реляционную SQL транзакцию. Альтернативные реализации темпоральных транзакций, при которых proxy-уровень отображает темпоральную SQL-транзакцию в несколько обычных SQL-транзакций, могут привести к неразрешимым проблемам, если во время выполнения получится так, что одна из реляционных SQL-транзакций будет зафиксирована, а вторая – нет, что должно означать неудачу всей темпоральной SQL-транзакции, а не отдельной ее части. Выполнить же откат транзакции, зафиксированной в базе данных, может оказаться просто невозможно, так как изменения уже могут быть использованы параллельно работающими транзакциями. Поэтому для поддержки параллельно работающих темпоральных транзакций необходимо реализовывать их в виде одиночных SQL-транзакций, иначе не удастся обеспечить поддержку ACID-свойств[22].

Кроме этого, недостаточно требовать только то, чтобы каждая темпоральная SQL транзакция отображалась в одну SQL-транзакцию. Для каждой транзакции необходимо обеспечить, чтобы в ее пределах транзакционное время было константно, но тогда встает вопрос, как его выбирать, ведь условия на специальное значение *СЕЙЧАС* должны проверяться с тем же значением константы. Здесь возможны два основных варианта: транзакционное время выбирается в самом начале или в самом конце выполнения транзакции. Первый вариант является некорректным, так как параллельная транзакция может внести изменения в таблицы, которые еще только понадобятся данной транзакции. Поэтому наиболее удачным выбором является момент времени непосредственно перед фиксацией транзакции, после того, как получены блокировки на все затрагиваемые объекты. Но и здесь возможны проблемы, если транзакция получит одно значение транзакционного времени, а будет зафиксирована чуть позже, так что между этими двумя моментами будет благополучно выполнена некорректная выборка. Подобная проблема может возникнуть и при параллельном выполнении двух различных транзакций.

Один из способов, который сможет гарантировать корректную поддержку ACID-свойств темпоральных транзакций, состоит в реализации дополнительного механизма блокировок на proxy-уровне. Отметим, что восстановление базы данных, которое является важной частью работы СУБД, является прозрачным для конечного пользователя. При использовании многоуровневой архитектуры для конечного пользователя промежуточный слой ничем не отличается от СУБД, поэтому при проектировании прослойки можно полностью положиться на механизмы восстановления, реализованные в СУБД. Также нет причин, по которым они стали бы работать быстрее или медленнее.

[22] Если, конечно, не реализовывать на proxy-уровне собственный механизм управления транзакциями над механизмом управления транзакциями базовой СУБД.

5.8. Эффективность при работе с темпоральной СУБД

Одним из наиболее важных вопросов при использовании баз данных является эффективность работы приложений с соответствующей СУБД. Для повышения скорости обработки запросов в реляционных СУБД традиционно применяются индексы для выборки данных из таблиц, а также статистики и различные эвристики при выборе алгоритма соединения данных из нескольких таблиц ([GJS+05]). При разработке темпоральных СУБД значительное внимание уделялось именно вопросам производительности, так как в общем случае небольшая доля «активных» данных из постоянно растущего объема информации в базе данных приводила к резкому спаду производительности при интенсивном использовании и/или недостаточном внимании при разработке приложения.

Наиболее очевидным, а также доступным способом оптимизации темпоральных приложений является использование индексов. Например, при добавлении дополнительных специальных столбцов для интервалов транзакционного времени необходимо включить верхний предел во все уникальные индексы. С другой стороны, необходимость постоянно разделять данные на «активные» и «устаревшие» требует их разделения на ранней стадии обработки, что также может быть обеспечено с помощью индексов. Аналогичная ситуация возникает при соединении темпоральных таблиц, так как оно часто проводится именно по специальным темпоральным столбцам.

Однако встает вопрос: должны ли темпоральные столбцы располагаться до или после остальных столбцов (обычного реляционного) индекса? Если они будут идти после всех обычных столбцов, то система получится почти «реляционной», так как большинство операций вначале будет выполняться именно на основе реляционных условий, а лишь затем будет применяться ограничение по времени, а значит, все проблемы с огромным объемом информации ложатся на традиционное реляционное ядро СУБД. Если же темпоральные столбцы будут располагаться перед обычными, то объем обрабатываемых данных будет напрямую зависеть лишь от наличия ограничений на темпоральные столбцы; например, при попытке получить историю отдельного кортежа при отсутствии темпоральых ограничений системе придется просмотреть историю всех кортежей (точнее, полностью весь индекс). Кроме того, подобный индекс будет довольно часто перестраиваться при изменениях данных темпоральных столбцов. С другой стороны, единственным изменением кортежа в таблице с темпоральной поддержкой транзакционного времени является установка вместо *СЕЙЧАС* точной верхней границы интервала (в момент удаления или изменения кортежа).

Описанные выше проблемы касались выборок из одной темпоральной таблицы. Наибольший же интерес представляют выборки из нескольких таблиц и соответствующее соединение результатов. До сих пор разрабатываются различные алгоритмы и предлагаются эвристики для

решения задачи эффективного соединения нескольких таблиц в традиционной реляционной СУБД. Частично они позволяют решить проблему производительности при соединении таблиц, но вся оптимизация возлагается на СУБД. Одновременно с этим следует помнить, что данные хранятся в интервальном представлении, поэтому соединение будет либо строиться на нестрогих неравенствах, либо алгоритм будет выполняться темпоральным ядром СУБД с учетом упаковки и распаковки темпоральных интервалов. Отсюда следует, что доступным путем является оптимизация преобразований различных темпоральных запросов в реляционные запросы и обратно с учетом построенных индексов. С другой стороны, в большинстве случаев разработчики приложений не смогут получить значительный выигрыш в эффективности, если будут сами формулировать реляционные запросы, а не пользоваться автоматическими преобразованиями, но первый путь сложнее и чреват появлением ошибок.

Таким образом, у создателей прототипов темпоральных СУБД не было широкого выбора методов оптимизации, а единственным действенным способом являлось расширение индексов или использование промежуточных алгоритмов, но за счет увеличения потока данных между groxy-уровнем и реляционной СУБД. При этом исследователи предлагали различные расширения стандартных подходов к индексированию реляционных баз данных и хранению данных.

6. Смежные и дополнительные области исследований

В предыдущем разделе рассматривались темпоральные базы данных в общем случае, когда не предполагалось наличие какой-либо дополнительной специфики в представляемой информации или ее формате. Однако существует много областей, в которых время связано с какими-нибудь конкретными характеристиками или показателями модели, и поэтому можно говорить не просто о темпоральной системе, а о некотором специфическом представлении в расчете на данную задачу. Далее мы рассмотрим подобные системы, а также выделим некоторые специфические форматы представления и хранения данных.

6.1. Темпоральные базы XML-данных

В начале этого века формат XML, по сути, стал стандартом представления данных для обмена в сети. Разработчики реляционных СУБД начали включать поддержку XML в свои продукты. Одновременно с этим многие исследователи темпоральных баз данных обратили свое внимание на возможность работы с темпоральными базами XML-данных. Первоначально большие надежды возлагались на создание темпоральной XML-СУБД, так как стандарт языка темпоральных запросов еще не был окончательно принят, а для его реализации потребовалось бы создание новых алгоритмов, оптимизаторов и т.п.; поэтому внесение темпоральной поддержки в ядро

XML-СУБД было вполне вероятным. Однако многие производители реляционных СУБД минимальными усилиями добавили в свои продукты поддержку баз XML-данных, а не стали создавать новые системы.

Имеется несколько работ, в которых предлагается темпоральное расширение языка XQuery [GS03], а также описываются темпоральные системы, предназначенные для хранения XML документов [GS03+]. В отличие от реляционной модели, в которой обычно используется плоское представление данных, в XML изначально применяется древовидное представление, поэтому появляется больше вариантов темпоральной поддержки. Например, метки времени могут являться наследуемыми атрибутами, что позволяет несколько упростить формулировку многих запросов и применить дополнительные способы оптимизации. С другой стороны, плоская структура является более «предсказуемой» при работе и позволяет гораздо проще можно получить многие оценки, необходимые оптимизатору. Темпоральные базы XML-данных используются в тех случаях, когда исходные данные являются плохо структурированными или уже представлены в формате XML.

6.2. Базы данных мультимедиа

Ранее говорилось о темпоральных базах данных, где каждый объект связан с некоторым моментом времени. Однако возможны ситуации, когда внутри самого объекта можно выделить некоторую линию времени, причем невозможно разбить объект на соответствующие части без нарушения его структуры и целостности. Простым примером могут служить базы данных мультимедиа, например, аудио и видео записей. Очень часто интерес представляет не только запись целиком, но и ее содержимое, привязанное к определенным моментам. Более того, не всегда возможно хранить дополнительную информацию непосредственно в самой записи, например, аннотации или субтитры. Поэтому можно выделить специальную разновидность темпоральных баз данных, где темпоральность проникает непосредственно в объект и тесно с ним связана.

6.3. Простанственно-временные базы данных

Время и пространство очень похожи, и их сложно игнорировать как некоторые характеристики объектов и событий. Причем время упоминается в разговоре о любом объекте или событии, а пространственные координаты практически всегда представляют интерес, когда речь идет о движущихся объектах. Если говорить о некоторой базе данных, которая хранит информацию о движущихся объектах, то ее статическое представление малоинтересно, поскольку основной интерес представляет именно изменение местоположения наблюдаемых объектов с течением времени. В подобных

системах невозможно игнорировать темпоральную составляющую, поэтому мы и приходим к понятию пространственно-временной[23] базы данных.

Примером пространственно-временной базы данных может служить информация о перемещениях и местонахождении, например, автомобилей, получаемая с помощью GPS-систем или датчиков на дорогах. Другой пример – отслеживание движений товаров на складах и между ними. Отметим, что пространственные данные сами по себе являются трудными для анализа, а при динамическом анализе объем данных очень сильно возрастает, и для создания эффективных пространственно-временных систем необходимы специальные алгоритмы. Хотя соответствующие исследования направлены на решение конкретных задач, обнаруживаемые методы часто можно использовать и в других типах темпоральных систем.

6.4. Механизм снимков состояний

Одним из частных случаев темпоральных баз данных является система с поддержкой снимков состояний. Механизм снимков состояний является самым простым способом реализации темпоральной поддержки, но приемлемым лишь в ограниченном количестве случаев. Снимок реляционной таблицы является независимой копией текущего состояния этой таблицы во время создания снимка. Соответственно, снимки получаются из базовых таблиц, но они не могут уже изменяться после создания, даже если меняются базовые таблицы.

Общий механизм снимков состояний может быть применен тремя различными способами. Во-первых, возможно ручное создание снимка состояния по специальной команде СУБД, например, *generate-snapshot*, которая создает снимок. Этот подход можно назвать созданием «ручного альбома». Второй подход состоит в автоматическом создании снимков через определенный пользователем промежуток времени; например, можно создавать снимок в конце каждого месяца и хранить его в течение года. Можно считать, что такой подход приводит к появлению «автоматического альбома». При третьем подходе снимки создаются последовательно по мере изменения базовой таблицы, то есть новый снимок создается каждый раз, когда обновляется состояние. Это уже можно назвать «съемкой фильма».

Подобный механизм может применяться для обычных таблиц, но нет никаких оснований, чтобы не применять его для событий, связанных с историей. Последовательные снимки таблицы (третий подход) приводят почти к обычной темпоральной таблице. При применении каждого из первых двух подходов получается темпоральная таблица специального вида.

[23] Здесь мы используем термин «пространственно-временные», а не «пространственно-темпоральные» базы данных, так как он подходит больше, а путаницы с ударением не возникает.

6.5. Ветвление линий времени и версии

Существует множество приложений, где требуется хранить и совместно использовать различные версии одного и того же объекта. С другой стороны, в эти версии могут вноситься любые изменения, и не требуется отслеживать их в системе. В таких случаях мы приходим к понятию версии. Версионность может поддерживаться как для отдельных строк таблицы, так и для набора таблиц базы данных. При этом должны существовать операции перехода к определенной версии, а также создания новой версии объекта. Главное отличие системы с поддержкой версий от обычной темпоральной системы состоит в том, что здесь могут иметься различные условия на соответствие версий при их объединении.

Еще одним вариантом темпоральных систем являются системы с ветвлением линий времени. В таких системах нельзя просто говорить о некотором моменте времени или об определенной версии, все это должно применяться относительно какого-то пути ветвления. То есть в некоторых точках на линии времени создаются развилки, которые продолжают существовать независимо друг от друга до того момента, пока не будет выбрано и зафиксировано какое-либо определенное продолжение. Подобные базы данных могут использоваться в системах принятия решений или при занесении данных, истинность которых точно не известна. Здесь не обязательно, чтобы заносимые данные были связаны со временем, так как, возможно, они будут просто иметь разные версии. Обратим внимание, что результат запроса не будет однозначным; поэтому система не может автоматически использоваться унаследованными приложениями, как это было с обычной темпоральной системой.

7. Предложения от производителей коммерческих СУБД

Выше уже говорилось, что не существует коммерческой реализации полноценной темпоральной СУБД, но многие производители предлагают различные расширения и дополнения, которые позволяют работать с темпоральными данными (или некоторым специальным их типом). Ниже представлены краткие характеристики подобных разработок.

7.1. TimeDB

Прототип темпоральной СУБД TimeDB был создан в 1997 г. Андреасом Стейнером в ходе работы по подготовке его диссертации [Ste98]. В прототипе поддерживалось как действительное, так и транзакционное время, но он был привязан к конкретной реляционной СУБД, так как в нем использовался интерфейс уровня вызовов компании Oracle. Позже была выпущена версия TimeDB 2.0, реализованная на языке Java. Для работы с реляционной СУБД использовался интерфейс JDBC, поэтому не было привязки к конкретному

производителю. Версию TimeDB 2.0 Beta 4 можно свободно скачать с сайта компании TimeConsult в исследовательских целях [TIM07]. Неизвестны дальнейшие пути развития проекта или какие-либо изменения в системе, произошедшие после 1999 года.

7.2. Informix TimeSeries Datablade

Модуль TimeSeries компании Informix предназначен для обработки и анализа динамики процессов на основе временны*х* рядов данных. Он содержит определение новых типов данных – временного ряда и календаря, а также предоставляет более сорока функций для обработки данных, содержащих временны*е* метки. Данный модуль предоставляет большие возможности по анализу динамики отдельного процесса, а также эффективному хранению данных и доступа к ним. Если ряд является регулярным, то доступ константен для любого момента времени. Здесь присутствует оптимизация вполне конкретного пути доступа, поэтому нельзя назвать TimeSeries Datablade полноценным темпоральным решением для действительного времени.

7.3. Immortal DB (прототип от Microsoft)

Целью проекта Immortal DB, реализованного исследовательским подразделением Microsoft, было предоставление поддержки транзакционного времени, встроенной в SQL сервер, а не основанной на каком-либо proxy-уровне ([LBM+05]). Была предпринята попытка продемонстрировать, что в СУБД, кроме поддержки снимков, может быть реализована полноценная поддержка транзакционного времени, причем с хорошей производительностью. При реализации поддержки был расширен стандартный механизм сномков состояния, позволяющий получить некоторый снимок базы данных на момент времени прошлого. Одна из проблем, стоявших перед разработчиками, состояла в обеспечении обновлений и модификации данных при минимальных дополнительных затратах, поэтому была введена поддержка разбиения файловых страниц базы данных при переполнении как по ключу, так и по временному атрибуту. Механизмы контроля параллельного доступа и восстановления были полностью интегрированы с соответствующими функциями MS SQL Server.

7.4. Технология Oracle Flashback – шаг к темпоральной СУБД

В Oracle9*i* появился механизм Flashback Query [Ora07a] – мощный, простой и полностью безопасный способ восстановления системы от ошибок пользователя. Он также позволяет пользователям без внесения каких-нибудь структурных изменений в базу данных просмотреть состояние базы данных на какой-либо момент в прошлом. Данная технология позволяет исправлять пользовательские ошибки. Операции Flashback Query могут выполняться без участия администратора, и это позволяет разработчикам добавлять в свои приложения функции восстановления данных. Для использования Flashback Query требуется включение автоматического управления восстановлением

(Automatic Undo Management) вместо использования сегментов отката (Rollback Segments). При этом в Flashback Query используются именно данные восстановления, поэтому его использование не сказывается на производительности.

В Oralce9*i* механизм Flashback Query позволял получить доступ только к некоторому статическому снимку данных. В Oracle10*g* к данной технологии добавились еще несколько средств: Flashback Versions Query позволяет получить вместо статической картинки «фильм» изменений, Flashback Transaction Query дает возможность увидеть изменения, внесенные заданной транзакцией, а средства восстановления Flashback Database, Flashback Query и Flashback Drop позволяют вернуться к предыдущему состоянию за константное время, не зависящее от объема базы данных.

Если подвести некоторый итог, то можно сказать, что СУБД Oracle стала уже почти темпоральной СУБД (с поддержкой транзакционного времени). Однако создание сколько-нибудь сложного запроса с использованием технологии Flashback опять полностью зависит от разработчика, так как никакой дополнительной поддержки множественных темпоральных операций не предусмотрено. Кроме того, даже для разрешенных темпоральных запросов существуют довольно жесткие ограничения.

7.5. Решение Oracle Workspace Manager – многоверсионность данных и поддержка снимков состояний

Кроме рассмотренной ранее технологии Oracle Flashback в Oracle 10g можно использовать механизм Oracle Workspace Manager – это одна из возможностей СУБД, позволяющая управлять текущими, предполагаемыми и историческими значениями данных в одной и той же базе данных [Ora07b]. Oracle Workspace Manager использует рабочие пространства в качестве виртуального окружения для изоляции совокупности изменений данных, хранения истории изменений данных и создания множественных сценариев развития для анализа возможного будущего. Создаваемые рабочие пространства могут наследоваться от других рабочих пространств или стандартного «актуального» рабочего пространства LIVE (по умолчанию). При этом фиксируется текущее состояние рабочего пространства-предка. В рамках конкретного рабочего пространства также можно использовать точки сохранения, которые позволяют откатывать изменения к определенному моменту прошлого. Также предусмотрены операции по слиянию рабочего пространства-потомка с рабочим пространством-предком. Отдельный интерес представляет опция полного отслеживания всех изменений данных, включая операции добавления, удаления и обновления. Фактически, это поддержка транзакционного времени. С точки зрения работы с темпоральным данными стоит отметить возможность зафиксировать запросы на указанный момент времени.

С технической точки зрения рабочие пространства представляют собой набор представлений базы данных с обработчиками триггеров *INSTEAD OF*. Когда

пользователь желает добавить для таблицы поддержку версионности, менеджер рабочих областей переименовывает таблицу в <имя таблицы>_LT, добавляя к ней четыре служебных столбца, после чего создает представление с названием исходной таблицы <имя таблицы>, для которого обработчики триггеров *INSTEAD OF* производят необходимые действия по изменению данных в исходных таблицах. Если немного упрощать, то данное решение является решением промежуточного слоя, только реализованного непосредственно внутри конкретной СУБД.

СУБД Oracle еще не стала полностью темпоральной СУБД, несмотря на наличие средства Oracle Workspace Manager, в котором была реализована часть идей SQL/Temporal. Однако прослеживаемая тенденция позволяет утверждать, что работы над созданием полноценной реализации темпоральной СУБД со стороны производителей коммерческих систем ведутся. Более того, для конкретных классов задач создаются отдельные темпоральные расширения, которые проходят «боевую» проверку на реальных задачах.

8. Актуальные вопросы и задачи, перспективы исследований

При описании темпоральных систем уже подчеркивалось наличие отдельных проблем, которые должны быть решены прежде, чем будет создана полноценная темпоральная СУБД. С другой стороны, опыт технологии Oracle Flashback показывает, что для конкретной задачи можно реализовать специализированную темпоральную поддержку. Кроме того, существует множество проблем и задач, которые могут быть решены в рамках исследований в области темпоральных и пространственно-временных баз данных [RHE+04]. В этом разделе перечисляются некоторые из подобных задач.

8.1. Эффективные пользовательские интерфейсы и представления

Созданные прототипы пространственных и темпоральных систем баз данных выявили существенные ограничения у существующих интерфейсов пользователя, основанных на окнах и меню. Поэтому приветствуются идеи по разработке новых способов взаимодействия пользователей с системой с использованием, например, световых указателей. Также требуется проведение исследований с целью нахождения эффективных методов визуализации пространственно-временных данных в контексте статических и анимированных графиков/карт.

8.2. Извлечение данных и знаний из пространственно-временных систем

Важными областями исследований являются извлечение информации и обнаружение знаний. Большинство работ может быть разделено на три категории:

- Извлечение темпоральных ассоциациативных правил, которые определяют зависимости в транзакционных и реляционных данных, обладающих темпоральным компонентом. В меньшем числе работ исследуются методы извлечения пространственных ассоциативных правил.
- Пространственная кластеризация с целью группировки схожих объектов в один кластер и разнесения различных объектов по разным кластерам. В данном случае схожесть определяется как пространственными, так и непространственными атрибутами объектов, а также любыми другими неоднородностями, которые могут присутствовать.
- Анализ временных последовательностей ст целью обнаружения часто встречающихся шаблонов в значениях атрибутов с течением времени.

Значительная часть этих исследований направлена на поиск семантики пространства и времени, дающей возможность использования алгоритмов извлечения знаний. Однако в большинстве случаев применяется либо пространственная, либо темпоральная семантика, а не их комбинация

8.3. Новый уровень мобильности

Относительно недавно появились беспроводные устройства для определения местонахождения и сети сенсоров. Это сделало возможным появление систем, поддерживающих мобильных пользователей способами, которые были невозможны ранее. В таких системах требуется поддержка пространственно-временных баз данных в условиях интенсивных потоков данных от беспроводных устройств.

8.4. Пространственное разрежение

Хотя стоимость дисковой памяти постоянно уменьшается, а ее объем увеличивается, остается потребность в удалении устаревших данных, которые больше не представляют интереса. Подобная проблема неоднократно исследовалась с позиций темпоральных баз данных, но важной задачей является развитие существующих методов и создание собственных алгоритмов для пространственных и пространственно-временных баз данных.

8.5. Нетрадиционные методы доступа

Для решения нетрадиционных проблем пространственно-временных баз данных требуются новые подходы. Например, во многих работах, посвященных движущимся объектам, для моделирования областей

пространства используются графы, а не евклидово представление пространства. Вместо евклидовых расстояний могут использоваться расстояния на дороге. Методы доступа к подобным данным должны соответствовать специфике проблемы. Исследования методов доступа к пространственно-временным данным, главным образом, фокусируются на двух аспектах: (1) хранение и поиск исторической информации и (2) предсказание будущего. Для решения первой задачи было предложено несколько индексных структур, минимизирующих объем хранимых данных и стоимость выполнения запроса. Подобные индексы обычно основываются на многоверсионных или трехмерных вариациях R-деревьев. Методы для предсказания будущего основаны на том предположении, что в дополнение к текущей позиции объектов известны и скорости их движения. Целью является нахождение объектов, которые будут удовлетворять пространственным условиям в некоторый момент (или интервал времени) будущего на основе заданных текущих скоростей движения (например, «на основе текущей информации найти все машины, которые будут в центре города через 10 минут»). Индексы, практически пригодные для предсказания будущего, основываются на TPR-деревьях и их вариациях (также являющих развитием идей R-деревьев).

Несмотря на огромное количество методов, которые явно фокусируются на выборку исторических данных или предсказание будущего, сейчас не существует ни одной индексной структуры, которая могла бы эффективно содействовать достижению обеих целей. Даже если бы существовала универсальная структура (например, многоверсионное TPR-дерево, сохраняющее всю предыдущую историю каждого объекта), она была бы неприменимой для некоторых приложений с интенсивным обновлением. Например, обновление (удаление или повторная вставка) TPR-дерева может привести к доступу более чем к 100 вершинам, и можно просто не успеть выполнить эту операцию до момента следующего обновления этого же объекта. Даже при небольшом числе движущихся объектов и небольшой скорости обновлений TPR-дерево (или любой другой индекс) не может «следить» за быстрыми изменениями данных. Поэтому для приложений с интенсивным обновлением кажутся более подходящими структуры данных в основной памяти, и в этой области необходимы дальнейшие исследования.

8.6. Новые типы пространственно-временных запросов

Существует множество областей, где могут быть эффективно использованы пространственно-временные базы данных, но для решения задач оказываются недостаточными возможности формулировки запросов языка SQL. К запросам нового типа относятся непрерывные запросы, где результат сильно зависит от темпорального контекста. Примером непрерывного запроса может служить следующий: «на основе текущего положения и скорости автомобиля найти две ближайшие АЗС в течение следующих пяти минут?» Результат в форме <{A, B}, [0,1)>, <{B,C}, [1,5)> означает, что АЗС A и B будут ближайшими в

интервале времени [0,1), а АЗС B и C – в интервале [1,5). Заметим, что соответствующий моментальный запрос («какие две АЗС являются сейчас ближайшими?») в высоко динамичном окружении обычно будет бессмысленным: если точка запроса или объект базы данных двигается, то результат может сразу стать недействительным.

В любом пространственном запросе имеется непрерывная составляющая, условие завершения которой зависит от потребностей пользователя или приложения. Рассмотрим, например, оконный запрос, где окно (и, возможно, объекты базы данных) двигается и/или изменяется с течением времени. Условием завершения может быть время (следующие пять минут), условие на результат (например, пока в окне запроса не останется только один объект, или пока результат не изменится три раза), условие на окно запроса (пока окно запроса не достигнет определенной точки в пространстве) и т.д.

Основным отличием от непрерывных запросов к традиционным базам данных является то, что в случае пространственно-временных баз данных для фиксации динамического поведения объекта не обязательно требуются обновления базы данных; поведение может сохраняться как функция от времени с использованием подходящих индексов. Кроме того, даже если объекты остаются статическими, результаты могут изменяться из-за динамической природы самого запроса (например, движущееся окно запроса), который также может быть представлен в виде функции от времени. Таким образом, пространственно-временной непрерывный запрос может быть вычислен немедленно (в текущий момент времени) с использованием параметризованной информации о динамическом поведении запроса или объектов базы данных, и будет выдано несколько результатов, каждый из которых относится к соответствующему интервалу времени в будущем.

8.7. Приближенные запросы

В некоторых пространственно-временных приложениях по причине большого объема данных и высокой скорости обновлений требуется приближенное вычисление запросов. Например, в системах управления движением трансорта исходные данные обычно представляются в виде потоков данных (например, через сенсоры, встроенные на дорожной сети), которые потенциально не ограничены в объеме. Поэтому нереальна материализация всех данных. Более того, даже если бы все данные были сохранены, точная обработка была бы слишком дорогой из-за больших размеров индекса, поскольку при использовании любого алгоритма выполнения запроса потребуется пройти в индексе полный путь от корня до листовой вершины. Наконец, для многих приложений требуется именно приближенная суммарная информация об объектах, удовлетворяющих определенному пространственно-временному предикату (например, «количество автомобилей в центре города в течение 10 минут»), а не точные данные об объектах (например, номера машин).

8.8. Неопределенности при обработке «неточных» данных

Неопределенность присуща большинству пространственно-временных приложений из-за ошибок в измерениях/оцифровке и отсутствия или неполноты информации. Допустим, например, что пользователь с карманным компьютером хочет узнать расстояние по шоссе до ближайшего ресторана. Хотя пользователь может находиться на некотором участке дороги, система может не суметь определить это из-за неточности GPS-приемника. В подобных ситуациях может задаваться допуск *dT*, такой что любая точка, удаленная от дороги на расстояние, не большее *dT*, считается находящейся на дороге. В качестве альтернативы можно привязать точку к ближайшей дороге, предполагая наличие неполноту информации (например, наличие незарегистрированного проезда улицу), или считать дорогу недоступной в зависимости от особенностей приложения. Похожие проблемы существуют для траекторий объектов, так как движение непрерывно, а измерения дискретны. При разработке приложений необходимо также учитывать возможность соединения нескольких таблиц, в каждой из которых данные могут быть неточны. В этом случае требуется разработка специальных индексов и оптимизация хранения информации, чтобы подобные запросы выполнялись эффективно, причем в результат может быть включен и показатель неопределенности.

9. Итоги и перспективы

Если посмотреть на ситуацию, сложившуюся в области исследований темпоральных баз данных, то можно отметить, что необходимость создания эффективных алгоритмов обработки и новых методов хранения данных является одной из важных задач, встающей во многих областях. С другой стороны, так и не было создано какое-либо универсальное решение в рамках расширения реляционной модели и стандарта языка запросов SQL. Однако отдельные разработки производителей коммерческих продуктов, а также решения для конкретных приложений вполне успешны. Поэтому в качестве одного из направлений исследований можно выделить совмещение темпоральной составляющей данных с другими характеристиками. Результаты подобных разработок приводят к расширению набора общих концепций, идей, методов и алгоритмов, связанных с анализом темпоральных данных и работе с темпоральными базами данных.

В дальнейшем набор проблем, стоящих перед исследователями, будет только расширяться, так как снижение цен на носители информации с одновременным увеличением их объема, а также повышение вычислительных возможностей систем приводят к тому, что можно проанализировать все больше и больше точных исторических данных, а не только совокупности некоторых статистик. Возможно, что в данный момент темпоральная технология находится близко к пику своего развития, и для дальнейших серьезных продвижений необходимы некоторые внешние события и факторы,

но остается очень много неисследованных проблем в смежных исследовательских и прикладных областях.

Литература

[AJS95] Arie Segev, Christian S. Jensen, and Richard T. Snodgrass. Report on The 1995 International Workshop on Temporal Databases. *ACM SIGMOD Record* 24(4), December 1995, http://acm.org/sigmod/record/issues/9512/temporal.ps

[BBJ+97] J. Bair, M. H. Böhlen, C. S. Jensen, and R. T. Snodgrass. Notions of Upward Compatibility of Temporal Query Languages. Wirtschaftsinformatik, 39(1):25–34, February 1997, http://www.cs.aau.dk/~csj/Thesis/pdf/chapter24.pdf

[BBJ98] M. H. Böhlen, R. Busato, and C. S. Jensen. Point-Versus Interval-Based Temporal DataModels. In Proceedings of the Fourteenth International Conference on Data Engineering, pp. 192–200, Orlando, Florida, February 1998, http://www.cs.aau.dk/~csj/Thesis/pdf/chapter7.pdf

[Böh95] Michael H. Böhlen. Temporal Database System Implementations. *ACM SIGMOD Record* 24(4), December 1995, http://www.sigmod.org/record/issues/9512/tdbms.ps

[BZ82] J. Ben-Zvi, "The Time Relational Model," PhD thesis, Computer Science Dept., UCLA, 1982

[CDI+97] J. Clifford, C. E. Dyreson, T. Isakowitz, C. .S. Jensen, and R. T. Snodgrass. On the Semantics of 'Now' in Databases. ACM Transactions on Database Systems, 22(2), June 1997, http://www.cs.aau.dk/~csj/Thesis/pdf/chapter4.pdf

[DDL02] C.J. Date, Hugh Darwen, Nikos Lorentzos. Temporal Data & the Relational Model. Morgan Kaufmann; 1st edition, November 19, 2002

[GJS+05] Dengfeng Gao, S. Jensen, T. Snodgrass, D. Soo. Join operations in temporal databases. The VLDB Journal, Volume 14 Issue 1, 2005, http://www.cs.aau.dk/~csj/Papers/Files/2004_gaoVLDBj.pdf

[GS03] Dengfeng Gao and Richard T. Snodgrass. Syntax, Semantics, and Query Evaluation of the τXQuery Temporal XML Query Language, *TimeCenter TR-72*, March 2003, http://oldwww.cs.aau.dk/research/DP/tdb/TimeCenter/TimeCenterPublications/TR-72.pdf

[GS03+] Dengfeng Gao and Richard T. Snodgrass. Temporal Slicing in the Evaluation of XMLQueries. Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003, http://www.vldb.org/conf/2003/papers/S19P03.pdf

[Jcc07] JCC's SQL Standards Page. http://jcc.com/SQL.htm JCC Consulting, 2007

[JT95] Jan Chomicki and David Toman. Implementing Temporal Integrity Constraints Using an Active DBMS. IEEE Transactions on Knowledge and Data Engineering 7(4), August 1995, http://www.cs.uwaterloo.ca/~david/papers-ride94.ps

[LBM+05] Lomet, D., Barga, R., Mokbel, M., Shegalov, G., Wang, R., and Zhu, Y. Immortal DB: Transaction Time Support for Sql Server. SIGMOD Conference, Baltimore, MD, June 2005, http://www-users.cs.umn.edu/~mokbel/ImmortalDBDemo.pdf

[Ora07a] Технология Oracle Flashback, http://www.oracle.com/technology/deploy/availability/htdocs/Flashback_Overview.htm, 2007.

[Ora07b] Oracle Workspace Manager, http://www.oracle.com/technology/products/database/workspace_manager/, 2007.

[RHE+04] Roddick, J. F., Hoel, E., Egenhofer, M. J., Papadias, D., and Salzberg, B. Spatial, temporal and spatio-temporal databases - hot issues and directions for phd research. *SIGMOD Rec.* 33(2), June 2004, http://acm.org/sigmod/record/issues/0406/RR1.SSTDpaper.pdf

[SAAB95] Richard T. Snodgrass, editor, Ilsoo Ahn, Gad Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Kaefer, Nick Kline, Krishna Kulkarni, T. Y. Cliff Leung, Nikos Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo and Suryanarayana M. Sripada, The TSQL2 Temporal Query Language, Kluwer Academic Publishers, 1995. Спецификация TSQL2 доступна по адресу ftp://ftp.cs.arizona.edu/tsql/tsql2/bookspec.pdf, а комментарии – на ftp://ftp.cs.arizona.edu/tsql/tsql2/eval.pdf

[SBJ96a] Snodgrass, R. T., M. H. Böhlen, C. S. Jensen and A. Steiner. Adding Valid Time to SQL/Temporal. ANSI X3H2-96-501r2, ISO/IEC JTC 1/SC 21/WG 3 DBL-MAD-146r2, November, 1996, http://www.cs.aau.dk/~csj/Thesis/pdf/chapter26.pdf

[SBJ96b] Snodgrass, R. T., M. H. Böhlen, C. S. Jensen and A. Steiner. Adding Transaction Time to SQL/Temporal, ANSI X3H2-96-502r2, ISO/IEC JTC 1/SC 21/WG 3 DBL-MAD-147r2, November, 1996, http://www.cs.aau.dk/~csj/Thesis/pdf/chapter27.pdf

[Sno96] Snodgrass, R.T. Addendum to Valid- and Transaction-time Proposals. ANSI X3H2-96-582, ISO/IEC JTC1/SC21/WG3 DBL MAD-203, November 1996, ftp://ftp.cs.arizona.edu/tsql/tsql2/sql3/ansi-96-582.pdf

[Sno97] Snodgrass, R.T. A Second Addendum to Valid- and Transaction-time Proposals. ANSI X3H2-97-010, ISO/IEC JTC1/SC21/WG3 DBL MAD-220, January 1997, http://www.informatik.uni-trier.de/~ley/db/systems/sql3.html

[Sno99] Richard T. Snodgrass. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann Publishers, 1999.

[Ste98] Andreas Steiner. A Generalisation Approach to Temporal Data Models and their Implementations. Doctoral Thesis, ETH No.12434, Department of Computer Science, ETH Zurich, Switzerland, 1998, http://www.timeconsult.com/Publications/diss.pdf

[Tim07] Официальный сайт проекта TimeDB, http://www.timeconsult.com/Software/Software.html

[TJS98] K. Torp, C. S. Jensen, and R. T. Snodgrass. Stratum Approaches to Temporal DBMS Implementation. In Proceedings of the 1998 International Database Engineering and Applications Symposium, Cardiff, Wales, UK, July 1998, http://www.cs.aau.dk/~csj/Thesis/pdf/chapter39.pdf

[Tom96] David Toman. Point vs. Interval-based Query Languages for Temporal Databases. Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, Montreal, Quebec, Canada, 1996, http://www.cs.uwaterloo.ca/~david/papers-pods96.ps

[Tom98] D. Toman. Point-Based Temporal Extensions of SQL and Their Efficient Implementation. Temporal Databases: Research and Practice, Springer; 1st edition, July 1, 1998, http://citeseer.ist.psu.edu/cache/papers/cs/3096/http:zSzzSzsolo.uwaterloo.cazSz~davidzSzpapers-dagstuhl97.pdf/point-based-temporal-extensions.pdf

[TS01] Paolo Terenziani, Richard T. Snodgrass. Reconciling Point-based and Interval-based Semantics in Temporal Relational Databases: A Proper Treatment of the Telic/Atelic Distinction, IEEE Transactions on Knowledge and Data Engineering, 16(5), May 2004, http://www.cs.arizona.edu/projects/stagg/papers/TR-60.pdf