Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего профессионального образования
«ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(национальный исследовательский университет)

На правах рукописи

ИВАНОВА Елена Владимировна

МЕТОДЫ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ СВЕРХБОЛЬШИХ БАЗ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ РАСПРЕДЕЛЕННЫХ КОЛОНОЧНЫХ ИНДЕКСОВ

Специальность 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:
СОКОЛИНСКИЙ Леонид Борисович,
доктор физ.-мат. наук, профессор

Челябинск – 2015

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

В настоящее время одним из феноменов, оказывающих существенное влияние на область технологий обработки данных, являются сверхбольшие данные. Согласно прогнозам аналитической компании IDC, количество данных в мире удваивается каждые два года и к 2020 г. достигнет 44 Зеттабайт, или 44 триллионов гигабайт [127]. Сверхбольшие данные путем очистки и структурирования преобразуются в сверхбольшие базы и хранилища данных. По оценкам корпорации Microsoft 62% крупных американских компаний имеют хранилища данных, объем которых превышает 100 Терабайт [61]. При этом современные технологии баз данных не могут обеспечить обработку столь крупных объемов данных. По оценке IDC в 2013 г. из всего объема существующих данных потенциально полезны 22%, из которых менее 5% были подвергнуты анализу. К 2020 году процент потенциально полезных данных может вырасти до 35%, преимущественно за счет данных от встроенных систем [127].

По мнению одного из ведущих специалистов мира в области баз данных М. Стоунбрейкера (Michael Stonebraker) для решения проблемы обработки сверхбольших данных необходимо использовать технологии СУБД [122]. СУБД предлагает целый спектр полезных сервисов, не предоставляемых файловой системой, включая схему (для управления семантикой данных), язык запросов (для организации доступа к частям базы данных), сложные системы управления доступом (грануляция данных), сервисы обеспечения согласованности данных (управление целостностью данных и механизм транзакций), сжатие (для уменьшения размера базы данных) и индексирование (для ускорения обработки запросов).

Для обработки больших данных необходимы высокопроизводительные вычислительные системы [67, 137]. В этом сегменте вычислительной

техники сегодня доминируют системы с кластерной архитектурой, узлы которых оснащены многоядерными ускорителями. Кластерные вычислительные системы занимают 87% списка TOP500 самых мощных суперкомпьютеров мира [128] (июнь 2015 г.). При этом в первой сотне списка более 50% систем оснащены многоядерными ускорителями. Самый мощный суперкомпьютер мира Tianhe-2 (Национальный суперкомпьютерный центр в Гуанчжоу, КНР) также имеет кластерную архитектуру и оснащен многоядерными ускорителями (сопроцессорами) Intel Xeon Phi. Его производительность составляет 33.9 PFLOP/S на тесте LINPACK, суммарный объем оперативной памяти – 1 петабайт. Недавние исследования показывают, что кластерные вычислительные системы могут эффективно использоваться для хранения и обработки сверхбольших баз данных [12, 14, 88, 103]. В разработке технологий параллельных систем баз данных [17, 124] к настоящему времени достигнут большой прогресс. Некоторый обзор научных публикаций по этой теме можно найти в [16, 137]. Однако в этой области остается целый ряд нерешенных масштабных научных задач, в первую очередь связанных с проблемой больших данных [122].

В последние годы основным способом наращивания производительности процессоров является увеличение количества ядер, а не тактовой частоты, и эта тенденция, вероятно, сохранится в ближайшем обозримом будущем [54]. Сегодня GPU (Graphic Processing Units) и Intel MIC (Many Integrated Cores) значительно опережают традиционные процессоры в производительности по арифметическим операциям и пропускной способности памяти, позволяя использовать сотни процессорных ядер для выполнения десятков тысяч потоков. Последние исследования показывают, что многоядерные ускорители могут эффективно использоваться для обработки запросов к базам данных, хранящимся в оперативной памяти [38, 117].

Оперативная память в качестве основного места хранения данных становится все более привлекательной в результате экспоненциального снижения отношения стоимость/размер [49, 59, 101]. Согласно отчету компании

Gartner, СУБД в оперативной памяти через 2-5 лет получат очень широкое распространение [86].

Одним из наиболее важных классов приложений, связанным с обработкой сверхбольших баз данных, являются хранилища данных [45, 58, 62, 98], для которых характерны запросы типа OLAP. Исследования показали [34, 36, 121], что для таких приложений выгодно использовать колоночную модель представление данных, позволяющую получить на порядок лучшую производительность по сравнению с традиционными системами баз данных, использующими строчную модель представления данных. Эта разница в производительности объясняется тем, что колоночные хранилища позволяют выполнять меньше обменов с дисками при выполнении запросов на выборку данных, поскольку с диска (или из основной памяти) считываются значения только тех атрибутов, которые упоминаются в запросе [24]. Дополнительным преимуществом колоночного представления является возможность использования эффективных алгоритмов сжатия данных, поскольку в одной колонке таблицы содержатся данные одного типа. Сжатие может привести к повышению производительности на порядок, поскольку меньше времени занимают операции ввода-вывода [25]. Недостатком колоночной модели представления данных является то, что в колоночных СУБД затруднено применение техники эффективной оптимизации SQL-запросов, хорошо зарекомендовавшей себя в реляционных СУБД. Кроме этого, колоночные СУБД значительно уступают строковым по производительности на запросах класса OLTP.

В соответствие с этим актуальной является задача разработки новых эффективных методов параллельной обработки сверхбольших баз данных в оперативной памяти на кластерных вычислительных системах, оснащенных многоядерными ускорителями, которые позволили бы совместить преимущества реляционной модели с колоночным представлением данных.

Степень разработанности темы

Проблематике систем баз данных посвящено большое количество работ, однако интерес к этой теме остается в научном сообществе по-прежнему высоким. Это связано прежде всего с феноменом больших данных, когда рост объемов информации, требующей хранения и обработки, опережает рост производительности серверных систем. Это стимулирует исследователей к поиску новых подходов к организации хранения и обработки больших данных. Среди российских исследователей наибольший вклад в развитие систем баз данных внесли научные группы, возглавляемые профессорами С.Д. Кузнецовым, Б.А. Новиковым, В. Э. Вольфенгагеным, С.В. Зыкиным. В области параллельных методов обработки баз данных значимые результаты получены научной группой под руководством профессора Л.Б. Соколинского. Среди зарубежных ученых, работающих в этой области, прежде всего следует упомянуть Майкла Стоунбрейкера (Michael Stonebraker), Дэвида ДеВитта (David DeWitt), Патрика Валдурица (Patrick Valduriez). В работах этих ученых и их коллег затрагиваются практически все аспекты современных исследований в области баз данных. Проблемам колоночной модели представления данных посвящены работы следующих ученых: Дэниэль Абади (Daniel Abadi), Самюэль Мэдден (Samuel Madden), Питер Бонц (Peter Boncz), Дон Батори (Don Batory), Джордж Копеланд (George Copeland), Хассо Платнер (Hasso Plattner) и др. Алгоритмы сжатия, ориентированные на хранение данных по столбцам, были исследованы в работах Дэниэля Абади (Daniel Abadi), Самюэля Мэддена (Samuel Madden), Мигеля Ферейры (Miguel Ferreira), Питера Бонца (Peter Boncz), Хассо Платнера (Hasso Plattner), Марцина Жуковского (Marcin Zukowski), Сандора Хемана (Sandor Heman), Нильса Неса (Niels Nes) и др. В последние годы большой интерес вызывали методы обработки баз данных с использованием многоядерных ускорителей (графиче-

ских ускорителей и многоядерных сопроцессоров). Этому аспекту посвящены работы Зилиана Зонга (Ziliang Zong), Брайана Ромсера (Brian Romoser), Самюэля Мэддена (Samuel Madden), Талала Бонни (Talal Bonny), Халеда Саламы (Khaled Salama), Симона Си (Simon See), Этикана Каруппиаха (Ettikan Karuppiah), Чиафенга Лина (Chiafeng Lin), Шьянминга Юана (Shyanming Yuan), Гюнтера Саака (Gunter Saake), Себастьяна Бресса (Sebastian Breß), Михаэля Шергера (Michael Scherger) и др. На сегодняшний день область научных исследований, связанная с применением многоядерных ускорителей к параллельной обработке сверхбольших данных на кластерных вычислительных системах с использованием колоночного представления информации, продолжает активно развиваться. Одной из важных нерешенных задач остается задача разработки методов параллельной обработки сверхбольших баз данных, сочетающих преимущества реляционной модели с колоночным представлением информации.

Цель и задачи исследования

Цель данной работы состояла в разработке и исследовании эффективных методов параллельной обработки сверхбольших баз данных с использованием колоночного представления информации, ориентированных на кластерные вычислительные системы, оснащенные многоядерными ускорителями, и допускающих интеграцию с реляционными СУБД. Для достижения этой цели необходимо было решить следующие *задачи*:

1. На основе колоночной модели хранения информации разработать вспомогательные структуры данных (колоночные индексы), позволяющие ускорить выполнение ресурсоемких реляционных операций.
2. Разработать методы фрагментации (распределения) колоночных индексов, минимизирующие обмены данными между вычислительными узлами при выполнении реляционных операций.

3. На основе использования распределенных колоночных индексов разработать методы декомпозиции основных реляционных операций, позволяющие организовать параллельное выполнение запросов без массовых пересылок данных между вычислительными узлами.
4. Реализовать предложенные модели и методы в виде колоночного сопроцессора СУБД, работающего на кластерных вычислительных системах с многоядерными ускорителями Intel Xeon Phi.
5. Провести вычислительные эксперименты, подтверждающие эффективность предложенных подходов.

Научная новизна

Научная новизна работы заключается в разработке автором оригинальной доменно-колоночной модели представления данных, на базе которой введены колоночные индексы с доменно-интервальной фрагментацией, и выполнением на ее основе декомпозиции основных операций реляционной алгебры. По сравнению с ранее известными методами параллельной обработки больших объемов данных предложенный подход позволяет сочетать эффективность колоночной модели хранения данных с возможностью использования мощных механизмов оптимизации запросов, разработанных для реляционной модели.

Теоретическая и практическая значимость работы

Теоретическая ценность работы состоит в том, что в ней дано формальное описание методов параллельной обработки сверхбольших баз данных с использованием распределенных колоночных индексов, включающее в себя доменно-колоночную модель представления данных. *Практическая ценность* работы заключается в том, что на базе предложенных методов и алго-

ритмов разработан колоночный сопроцессор для кластерной вычислительной системы с многоядерными ускорителями, позволяющий во взаимодействии с СУБД PostgreSQL получить линейное ускорение при выполнении ресурсоемких реляционных операций. Результаты, полученные в диссертационной работе, могут быть использованы для создания колоночных сопроцессоров для других коммерческих и свободно распространяемых реляционных СУБД.

Методология и методы исследования

Методологической основой диссертационного исследования является теория множеств и реляционная алгебра. Для организации параллельной обработки запросов использовались методы горизонтальной фрагментации отношений и методы организации параллельных вычислений на основе стандартов MPI и OpenMP. При разработке колоночного сопроцессора применялись методы объектно-ориентированного проектирования и язык UML.

Положения, выносимые на защиту

На защиту выносятся следующие новые научные результаты:

1. Разработана доменно-колоночная модель представления данных, на базе которой выполнена декомпозиция основных реляционных операций с помощью распределенных колоночных индексов.
2. Разработаны высокомасштабируемые параллельные алгоритмы выполнения основных реляционных операций, использующие распределенные колоночные индексы.
3. Выполнена реализация колоночного сопроцессора для кластерных вычислительных систем.

4. Проведены вычислительные эксперименты, подтверждающие эффективность предложенных подходов.

Степень достоверности результатов

Все утверждения, связанные с декомпозицией реляционных операций, сформулированы в виде теорем и снабжены строгими доказательствами. Теоретические построения подтверждены тестами, проведенными в соответствии с общепринятыми стандартами.

Апробация результатов исследования

Основные положения диссертационной работы, разработанные модели, методы, алгоритмы и результаты вычислительных экспериментов докладывались автором на следующих международных научных конференциях:

1. 38-я международная научная конференция по информационно-коммуникационным технологиям, электронике и микроэлектронике MIPRO’2015, 25-29 мая 2015 г., Хорватия, г. Опатия.

2. Международная научная конференция «Суперкомпьютерные дни в России», 28-29 сентября 2015 г., Москва.

3. Международная научная конференция «Параллельные вычислительные технологии (ПаВТ'2015)», 30 марта - 3 апреля 2015 г., Екатеринбург.

4. Международная научная конференция «Параллельные вычислительные технологии (ПаВТ'2014)», 1–3 апреля 2014 г., Ростов-на-Дону.

5. Международная суперкомпьютерная конференция «Научный сервис в сети Интернет: многообразие суперкомпьютерных миров», 22-27 сентября 2014 г., Новороссийск.

Публикации соискателя по теме диссертации

Основные результаты диссертации опубликованы в следующих научных работах.

Статьи в журналах из перечня ВАК

1. *Иванова Е.В. Соколинский Л.Б.* Колоночный сопроцессор баз данных для кластерных вычислительных систем // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 4. С. 5-31.
2. *Иванова Е.В. Соколинский Л.Б.* Использование сопроцессоров Intel Xeon Phi для выполнения естественного соединения над сжатыми данными // Вычислительные методы и программирование: Новые вычислительные технологии. 2015. Т. 16. Вып. 4. С. 534-542.
3. *Иванова Е.В. Соколинский Л.Б.* Параллельная декомпозиция реляционных операций на основе распределенных колоночных индексов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 4. С. 80-100.

Статьи в изданиях, индексируемых в SCOPUS и Web of Science

4. *Ivanova E., Sokolinsky L.* Decomposition of Natural Join Based on Domain-Interval Fragmented Column Indices // Proceedings of the 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO. IEEE, 2015. P. 223-226.

Статьи в изданиях, индексируемых в РИНЦ

5. *Иванова Е.В., Соколинский Л.Б.* Использование сопроцессоров Intel Xeon Phi для выполнения естественного соединения над сжатыми данными // Суперкомпьютерные дни в России: Труды международной конференции (28-29 сентября 2015 г., г. Москва). М.: Изд-во МГУ, 2015. С. 190-198.

6. *Иванова Е.В. Соколинский Л.Б.* Использование распределенных колоночных индексов для выполнения запросов к сверхбольшим базам данных // Параллельные вычислительные технологии (ПаВТ'2014): труды международной научной конференции (1–3 апреля 2014 г., г. Ростов-на-Дону). Челябинск: Издательский центр ЮУрГУ, 2014. С. 270–275.

7. *Иванова Е.В., Соколинский Л.Б.* Декомпозиция операции группировки на базе распределенных колоночных индексов // Наука ЮУрГУ. Челябинск: Издательский центр ЮУрГУ, 2015. С. 15-23.

8. *Иванова Е.В. Соколинский Л.Б.* Декомпозиция операций пересечения и соединения на основе доменно-интервальной фрагментации колоночных индексов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 1. С. 44-56.

9. *Иванова Е.В.* Исследование эффективности использования фрагментированных колоночных индексов при выполнении операции естественного соединения с использованием многоядерных ускорителей // Параллельные вычислительные технологии (ПаВТ'2015): труды международной научной конференции (30 марта - 3 апреля 2015 г., г. Екатеринбург). Челябинск: Издательский центр ЮУрГУ, 2015. С. 393–398.

10. *Иванова Е.В.* Использование распределенных колоночных хеш-индексов для обработки запросов к сверхбольшим базам данных // Научный

сервис в сети Интернет: многообразие суперкомпьютерных миров: Труды Международной суперкомпьютерной конференции (22-27 сентября 2014 г., Новороссийск). М.: Изд-во МГУ, 2014. С. 102-104.

В работах 1 – 8 научному руководителю Соколинскому Л.Б. принадлежит постановка задачи, Ивановой Е.В. – все полученные результаты.

Структура и объем работы

Диссертация состоит из введения, четырех глав, заключения и библиографии. В приложении 1 приведены основные обозначения, используемые в диссертации. Приложение 2 содержит сводку по операциям расширенной реляционной алгебры. Объем диссертации составляет 143 страницы, объем библиографии – 143 наименования.

Содержание работы

Первая глава, «Современные тенденции в развитии аппаратного обеспечения и технологий баз данных», посвящена обзору научных исследований по актуальным направлениям развития современных систем баз данных. Особое внимание уделяется методам обработки баз данных на кластерных системах с многоядерными ускорителями и колоночной модели хранения данных. Дается обзор публикаций, наиболее близко относящихся к теме диссертации.

Во второй главе, «Доменно-колоночная модель», строится формальная доменно-колоночная модель представления данных. Вводятся колоночные индексы. Описывается оригинальный способ фрагментации колоночных индексов, названный доменно-интервальной фрагментацией. Вводится понятие транзитивной фрагментации одного колоночного индекса относительно

другого для атрибутов, принадлежащих одному и тому же отношению. Рассматриваются методы декомпозиции реляционных операций на основе использования фрагментированных колоночных индексов.

В третьей главе, «Колоночный сопроцессор КСОП», описывается процесс проектирования и реализации программной системы «Колоночный сопроцессор КСОП» для кластерных вычислительных систем, представляющей собой резидентную параллельную программу, взаимодействующую с реляционной СУБД. Специфицируются ключевые функции колоночного сопроцессора. Кратко описываются алгоритмы создания, модификации и использования распределенных колоночных индексов для выполнения ресурсоемких операций над базой данных.

В четвертой главе, «Вычислительные эксперименты», приводятся результаты вычислительных экспериментов по исследованию эффективности разработанных в диссертации моделей, методов и алгоритмов обработки сверхбольших баз данных с использованием распределенных колоночных индексов.

В заключении в краткой форме излагаются итоги выполненного диссертационного исследования, представляются отличия диссертационной работы от ранее выполненных родственных работ других авторов, даются рекомендации по использованию полученных результатов и рассматриваются перспективы дальнейшего развития темы.

В приложении 1 приводятся основные обозначения, используемые в диссертационной работе.

В приложении 2 приводится список операций расширенной реляционной алгебры, используемых в диссертационной работе.

ГЛАВА 1. СОВРЕМЕННЫЕ ТЕНДЕНЦИИ В РАЗВИТИИ АППАРАТНОГО ОБЕСПЕЧЕНИЯ И ТЕХНОЛОГИЙ БАЗ ДАННЫХ

В данной главе рассматриваются тенденции развития аппаратного обеспечения и дается обзор научных исследований в области современных технологий баз данных. Особое внимание уделяется методам обработки баз данных на вычислительных системах с многоядерными ускорителями и колоночной модели хранения данных. Анализируются публикации, наиболее близко относящихся к теме диссертации.

1.1. Переход к многоядерным процессорам

В 1965 году соучредитель корпорации Intel Г. Мур сделал предсказание о возрастающей сложности интегральных микросхем в полупроводниковой промышленности [116]. Это предсказание стало известно как закон Мура. Закон Мура гласит, что число транзисторов на одной микросхеме удваивается примерно каждые два года. В действительности, производительность центральных процессоров удваивается в среднем каждые 21 месяц [110]. До недавнего времени увеличение производительности процессоров в определяющей мере обеспечивалось увеличением тактовой частоты, которая росла в геометрической прогрессии в течение почти 30 лет, но с 2002 года рост прекратился [101]. Потребляемая мощность, тепловыделение, а также скорость света стали ограничивающим фактором для закона Мура [93]. Дальнейшее повышение производительности процессоров связано с их многоядерностью. Для потребительского рынка многоядерные процессоры были выпущены в 2005 году, начиная с двух ядер на одном чипе, например, Advanced Micro Devices (AMD) Athlon 64 X2. На форуме разработчиков осенью 2006 года, корпорация Intel представила прототип 80-ядерного процессора, в то время как IBM представила в том же году Cell Broadband Engine, с десятью ядрами

[66]. В 2007 году компания Tilera представила Tile64 – многоядерный процессор для рынка встраиваемых систем, который состоял из 64 ядер [11].

Следующим этапом перехода к многоядерным процессорам стало использование в высокопроизводительных вычислениях многоядерных ускорителей. Первым суперкомпьютером с многоядерными ускорителями стала система IBM RoadRunnder [81], построенная с использованием процессоров с архитектурой Cell [44]. В июне 2010 на втором месте в списке TOP500 появился суперкомпьютер Dawning Nebulae [123] с производительностью 1.3 Petaflops в котором в качестве многоядерных ускорителей были использованы графические процессоры NVIDIA Fermi [96] с архитектурой CUDA. В 2012 году корпорация Intel выпустила вычислительный кластер Discovery [75], в котором в качестве многоядерных ускорителей были использованы сопроцессоры Intel Xeon Phi с архитектурой MIC [125]. Описание архитектура NVIDIA CUDA можно найти в работах [95, 138]. Далее в этом разделе кратко описывается архитектура Intel MIC, для которой создан сопроцессор баз данных, разработанный в рамках диссертационной работы.

В 2012 году корпорация Intel анонсировала новую линейку сопроцессоров Intel Xeon Phi с архитектурой Intel Many Integrated Core (Intel MIC), предназначенных для массивно-параллельных вычислений [77, 134]. Первое поколение сопроцессоров Intel Xeon Phi было построено по архитектуре KNights Corner (KNC). В соответствии с этой архитектурой сопроцессор включает в себя более 50 процессорных ядер семейства x86, каталоги тегов (Tag Directory, TD), контроллеры памяти GDDR5 (Memory Controllers, MC), объединенные высокоскоростным двунаправленным кольцом (см. **рис. 1**). Все указанные компоненты подключаются к кольцу через кольцевые шлюзы (Ring Stops, RS). Кольцевые шлюзы отвечают за передачу сообщений по кольцу, включая захват и освобождение кольца.

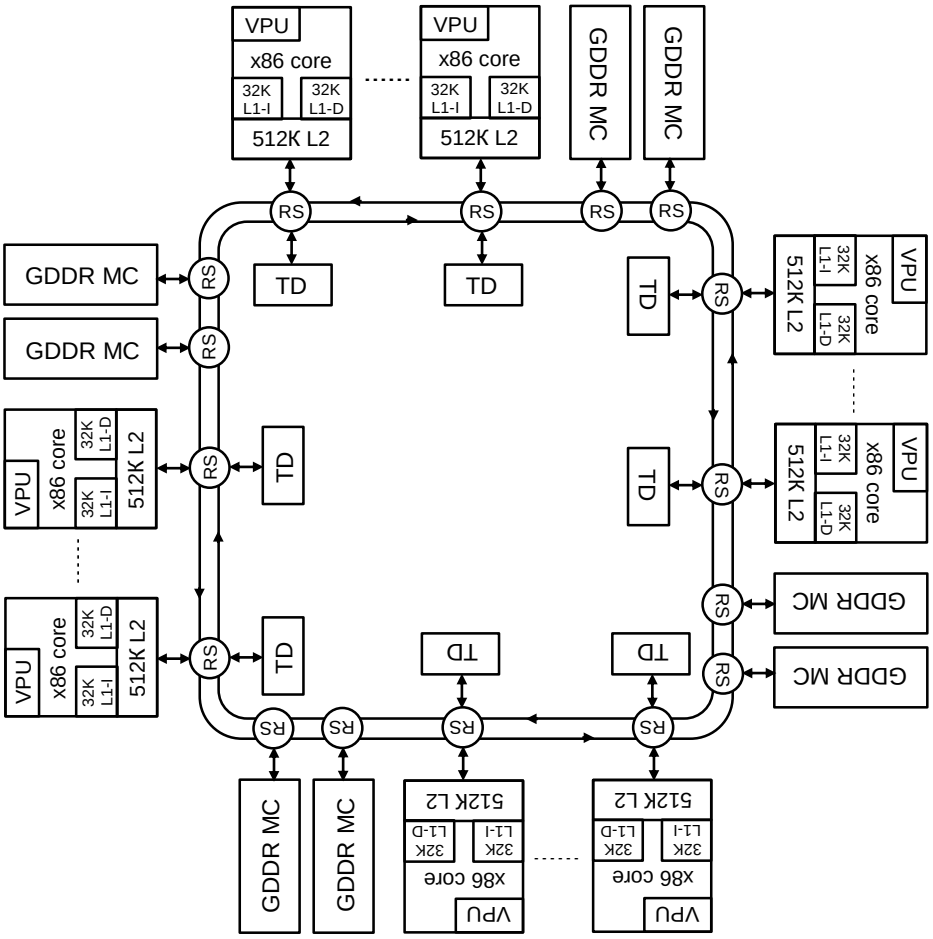


Рис. 1. Архитектура Knights Corner.

Каждое ядро содержит 4 компоненты: синхронный двухканальный конвейер, поддерживающий выборку и декодирование инструкций из 4 потоков команд; 512 разрядное SIMD устройство, называемое модулем векторной обработки (Vector Processing Unit, VPU); кэш первого уровня объемом 32 Кб для команд (L1-I) и 32 Кб для данных (L1-D); и кэш второго уровня объемом 512 Кб с полной поддержкой когерентности. Аппаратная поддержка когерентности кэшей второго уровня осуществляется с помощью каталогов тэгов (TD). Каждый каталог тегов реагирует на кэш-промахи, соответствующие определенной области памяти. Как видно на **рис. 1** пары контроллеров памяти чередуются с парами процессорных ядер. Адресное пространство памяти равномерно распределено по кольцу между всеми контроллерами па-

мяти и каталогами тэгов. Такой дизайн обеспечивает равномерное распределение трафика по кольцу, что является важным для обеспечения высокой пропускной способности шины.

При большом количестве ядер и когерентных кэшей стоимость кэш-промахов в KNC сравнима со стоимостью кэш-промахов в обычном процессоре x86. При возникновении промаха в кэше второго уровня, соответствующему каталогу тэгов посылается адрес запрашиваемой ячейки памяти. В зависимости от того, найден или не найден запрашиваемый адрес в кэше другого ядра, запрос пересылается этому ядру или соответствующему контроллеру памяти, и необходимые данные передаются по кольцу. Стоимость каждой пересылки данных по кольцу пропорциональна расстоянию между источником и получателем и в худшем случае может составлять сотни тактов. В среднем, латентность кэш-промахов в KNC может быть на порядок выше по сравнению с обычными многоядерными процессорами.

Сопроцессоры Intel Xeon Phi устанавливаются на отдельной плате, которая соединяется с центральным процессором Intel Xeon через шину PCI Express, по которой данные передаются от центрального процессора к плате ускорителя и в обратном направлении.

Для разработки программ для платформы Intel Xeon Phi используются стандартные языки программирования, такие как C, C++ и Fortran, а также стандартные технологии параллельной обработки данных: OpenMP, Intel MPI, Intel TBB, Intel Cilk, функции библиотеки Intel MKL.

В системе с Intel Xeon Phi программы могут исполняться в следующих режимах.

– В *симметричном режиме* (*symmetric mode*) программа компилируется в два исполняемых файла для запуска на центральном процессоре и сопроцессоре.

– В *режиме разгрузки* (*offload mode*) часть кода программы, выполняемой на центральном процессоре, передается для выполнения на сопроцессор. При этом объявленные переменные и структуры данных (все или частично) передаются из памяти основного процессора в память сопроцессора. После завершения работы сопроцессора данные копируются обратно в память основного процессора.
– В *естественном режиме* (*native mode*) программа запускается непосредственно на сопроцессоре, центральный процессор в вычислениях не участвует.

1.2. Обработка запросов с использованием многоядерных ускорителей

В данном разделе рассматриваются исследования по применению многоядерных ускорителей для обработки запросов к базам данных.

Большое количество работ посвящено использованию в обработки запросов графических процессорных устройств (ГПУ) [40]. В работах [65, 69] сравнивалась производительность обычных центральных процессорных устройств (ЦПУ) и ГПУ при выполнении запросов к базе данных. Было показано, что соединение на ГПУ выполняется в 2-7 раз быстрее, чем на ЦПУ, а выборка в 2-4 раза медленнее. Последнее объясняется тем, что затраты на передачу данных в ГПУ при выборке значительно превосходят затраты на вычисления. Тот же самый эффект наблюдается и для некоторых других реляционных операций. Был сделан вывод, что для того, чтобы достичь высокой производительности на ГПУ, необходимо минимизировать объемы передаваемых данных. В [38, 68] было показано, что ГПУ хорошо подходят для легко распараллеливаемых простых операций (вычисление предикатов, арифметические операции), в то время как ЦПУ много более эффективны при выполнении операций, включающих сложные структуры управления

или интенсивные обмены данными между потоками (нитями управления). Определение оптимального процессорного устройства для выполнения той или иной операции при обработке запроса в общем случае оказывается нетривиальной задачей.

В работах [42, 114] говорится о том, что современные процессорные устройства становятся все более разнородными, и для достижения оптимальной производительности СУБД должна включать в себя реализацию операций физической алгебры для множества разнородных процессорных устройств, что ведет к экспоненциальному увеличению вариантов исходного кода и росту стоимости разработки и сопровождения системы.

Хе (He) и другие исследователи показали [68], что в случае, когда база данных храниться на диске, использование ГПУ оказывается малоэффективным, так как в этом случае шина ввода-вывода становиться узким местом. Аналогичная картина наблюдается в случае использования сопроцессоров Intel Xeon Phi. В соответствии с этим при использовании ГПУ необходимо ориентироваться на базы данных, хранящиеся в оперативной памяти [48, 117].

Гходсниа (Ghodsnia) [60], Баккум (Bakkum) и другие исследователи [30] сравнивают строчные и колоночные хранилища в контексте эффективности использования ГПУ при выполнении запросов класса OLAP. Делается вывод, что колоночное хранилище является более предпочтительным в силу следующих трех причин: (1) возможно использовать коалесцированный доступ к памяти в ГПУ; (2) достигается больший коэффициент сжатия информации, что важно в силу ограниченности памяти ГПУ; (3) сокращается объем данных, необходимых для получения результирующего отношения. В настоящее время, практически все исследователи пришли к единому мнению, что ГПУ-ориентированные СУБД должны использовать колоночную модель представления данных и хранить базу данных в оперативной памяти [35].

Бресс (Breß) с соавторами [39] исследует эффективность различных моделей выполнения запросов на ГПУ. Известно два основных подхода: итераторный [63] и материализационный (bulk processing) [91]. Итераторный подход предполагает, что со всеми операциями физической алгебры в плане выполнения запроса связывается виртуальная функция next, которая при каждом вызове возвращает очередной кортеж промежуточного результирующего отношения. Таким образом организуется покортежный (конвейерный) режим выполнения операций физической алгебры. Основным преимуществом данного подхода является то, что промежуточные данные занимают минимум места. Однако этот подход имеет следующие существенные недостатки: синхронное покортежное выполнение операций, входящих в план запроса, приводит к тому, что все операции работают со скоростью самой медленной операции; увеличивается количество кэш-промахов; при выполнении промежуточных операций невозможно использовать внешний ускоритель, требующий в качестве входного параметра полный столбец данных. Материализационный подход позволяет эффективно кэшировать данные, минимизируя количество кэш-промахов. Кроме этого, он позволяет подключать внешние ускорители в любой точке плана.

В [39] было показано, что покортежное выполнение запросов неприемлемо при использовании ГПУ, так как в этом случае межъядерные коммуникации становятся узким местом. В соответствии с этим для ГПУ должна использоваться материализационная модель. Для уменьшения потерь, связанных с сохранением промежуточных отношений, в работах [48, 94, 133] был предложен подход, при котором в результате динамической компиляции в ходе выполнения запроса строится «мега»-оператор, выполняющей сразу несколько операций физической алгебры или даже весь запрос в целом. В этом случае отпадает необходимость материализовывать промежуточные отношения. Кортежи передаются между операциями через регистры или через общую память.

В [60] предложен подход, при котором вся база данных хранится в памяти ГПУ. Это позволяет существенно сэкономить время на передаче данных из общей памяти ЦПУ в память ГПУ. Более того, поскольку пропускная способность памяти ГПУ примерно в 16 раз выше пропускной способности шины PCI, применяя этот подход, можно получить еще большее ускорение. В качестве минусов можно указать следующее: (1) память ГПУ примерно на два порядка меньше, чем память ЦПУ, поэтому необходимо базу данных фрагментировать по многим ГПУ, сжимая при этом отдельные фрагменты [56]; (2) при хранении всей базы данных в ГПУ нельзя использовать ЦПУ, что существенно снижает потенциальную производительность системы. Для преодоления этого недостатка в [41, 68, 99] предлагается использовать гибридный подход, при котором часть базы данных хранится в оперативной памяти, а часть в памяти ГПУ. Операции по выполнению плана запроса также делятся между ЦПУ и ГПУ. Недостатком этого подхода является то, что ЦПУ и ГПУ не являются симметричными устройствами и, следовательно, операции между ними нельзя делить произвольным образом, а это приводит, в свою очередь, к существенному усложнению процесса разработки СУБД.

К настоящему моменту разработано достаточно большое количество ГПУ-ориентированных СУБД, среди которых можно отметить следующие: CoGaDB [37, 41], GPUDB [140], GPUQP [55, 68], GPUTx [70], MapD [92], Ocelot [71], OmniDB [139], Virginian [30]. В [126] рассматриваются принципы проектирования СУБД на ГПУ для обработки в реальном времени больших потоков данных, поступающих из внешних источников (например, из сенсорных сетей).

Проблематика исследований по созданию СУБД, ориентированных на архитектуру MIC, во многом сходна с исследованиями ГПУ-ориентированных СУБД. Одной из первых MIC-ориентированных СУБД является SQLPhi,

разработанная для сопроцессоров Intel Xeon Phi [97]. SQLPhi преобразует запрос на языке SQL [13] в последовательность псевдокоманд подобно тому, как это делается в SQLite [29]. Затем эти команды выполняются интерпретатором, написанным для Intel Xeon Phi. База данных в SQLPhi представляется в виде одного непрерывного файла и организуется в виде таблетов (tablets), каждый из которых хранит часть таблиц базы данных в некоторой фиксированной области диска. Структурно каждый таблет делится на три области: область метаданных, область первичных ключей и область данных. Данные представляются в колоночном виде. Эксперименты с SQLPhi показали, что без учета времени передачи данных в память Xeon Phi сопроцессор показывает лучшее время выполнения запроса по сравнению с центральным процессором. Однако затраты на передачу данных в память Xeon Phi оказываются узким местом и по общему времени Xeon Phi проигрывает центральному процессору примерно в 7 раз.

В работе [117] излагаются некоторые начальные идеи по построению параллельной СУБД для Xeon Phi. Базу данных предполагается хранить в оперативной памяти Xeon Phi. Предлагается «планочная» модель хранения. Каждому ядру Xeon Phi в памяти соответствует своя «горизонтальная» планка (fold). Кортежи отношения записываются вертикально, по одному кортежу на планку. Ели планок не хватает, то начинается заполняться второй вертикальный ряд, и так далее. Работа не дает ответов, как база данных делиться между различными процессорными узлами, кроме этого, предлагаемые решения не подтверждены реализацией и вычислительными экспериментами.

В [78] проводятся экспериментальные исследования хеш-соединения в памяти Intel Xeon Phi. Рассматриваются два вида алгоритмов: аппаратно-зависимые и аппаратно-независимые. На основе большого числа экспериментов делается вывод, что тонкая настройка под архитектуру Xeon Phi алгорит-

мов соединения обоих видов способна существенно увеличить производительность. Кроме этого показано, что во многих случаях аппаратно-независимый алгоритм не уступает и даже превосходит по производительности аппаратно-зависимый. Это, в частности, наблюдается в следующих трех случаях: 1) кортежи имеют большой размер; 2) отношение перекошено; 3) размер отношения очень мал. Следует отметить, что все эксперименты проводились на одном процессорном узле.

1.3. Колоночная модель хранения данных

В последнее время большой интерес у исследователей вызывают системы баз данных с хранением баз данных по столбцам. Такое представление данных называется *колоночным* [19, 22, 23, 100]. Колоночное представление в отличие от традиционного строкового представления оказывается намного более эффективным при выполнении запросов класса OLAP [24]. Это объясняется тем, что при выполнении запросов колоночная СУБД считывает с диска только те атрибуты, которые необходимы для выполнения запроса, что сокращает объем операций ввода-вывода и, как следствие, уменьшает время выполнения запроса. Недостатком колоночного представления является низкая эффективность при выполнении стоково-ориентированных операций, таких, например, как добавление или удаление кортежей. Вследствие этого колоночные СУБД могут проигрывать по производительности строковым при выполнении запросов класса OLTP.

В колоночном хранилище каждое отношение базы данных физически разбивается на столбцы (колонки), которые хранятся отдельно друг от друга. Раздельное хранение колонок позволяет колоночной СУБД считывать с диска только те колонки, которые соответствуют атрибутам, задействованным в запросе, в то время как строчная СУБД всегда считывает кортежи целиком. Это позволяет значительно экономить время на операциях

Рис. 2. Физическая организация колоночных и строчных хранилищ.

ввода-вывода. Аналогичные преимущества колоночные СУБД получают при копировании данных из основной памяти в регистры. Помимо этого, колоночные СУБД позволяют эффективно использовать ряд технических приемов, недоступных или неэффективных для строчных СУБД.

Первоначально колоночные хранилища были реализованы в ряде «академических» СУБД, среди которых следует упомянуть MonetDB [73], VectorWise (первоначальное название MonetDB/X100) [36] и C-Store [121]. Одной из первых коммерческих колоночных СУБД стала SybaseIQ [90]. Академические колоночные СУБД VectorWise и C-Store позднее эволюционировали в коммерческие системы Ingres VectorWise [142] и Vertica [82] соответственно. Начиная с 2013 года все основные производители СУБД включили в линейку продуктов колоночные версии своих систем: IBM [31], Microsoft [83, 84, 85], SAP [57] и Oracle [135]. В качестве современных колоночных СУБД также можно отметить EXASOL [20, 21, 53], Actian Vector [28], InfoBright [119] и SAND [115].

На рис. 2 схематично изображено основное отличие в физической организации колоночных и строчных хранилищ [23]: показаны три способа представления отношения Sales (Продажи), содержащего пять атрибутов. При колоночно-ориентированном подходе (рис. 2 (a) и (b)) каждая колонка хранится независимо как отдельный объект базы данных. Поскольку данные

в СУБД записываются и считываются поблочно, колоночно-ориентированный подход предполагает, что каждый блок, содержащий информацию из таблицы продаж, включает в себя данные только по некоторой единственной колонке. В этом случае, запрос, вычисляющий, например, число продаж определенного продукта за июль месяц, должен получить доступ только к колонкам prodid (идентификатор продукта) и date (дата продажи). Следовательно, СУБД достаточно загрузить в оперативную память только те блоки, которые содержат данные из этих колонок. С другой стороны, при строчно-ориентированном подходе (рис. 2 (c)) существует единственный объект базы данных, содержащий все необходимые данные, то есть каждый блок на диске, содержащий информацию из таблицы Sales, включает в себя данные из всех колонок этой таблицы. В этом случае отсутствует возможность выборочно считать конкретные атрибуты, необходимые для конкретного запроса, без считывания всех остальных атрибутов отношения. Принимая во внимание тот факт, что затраты на обмены с диском (либо обмены между процессорным кэшем и оперативной памятью) являются узким место в системах баз данных, а схемы баз данных становятся все более сложными с включением широких таблиц с сотнями атрибутов, колоночные хранилища способны обеспечить существенный прирост в производительности при выполнении запросов класса OLAP.

Первые исследования колоночного представления данных были выполнены в работах [33, 47, 79, 89]. Основные принципы, заложенные в этих работах, остаются справедливыми по настоящее время. Современные исследования в этом направлении концентрируются на разработке новых технических решений по повышению производительности СУБД, учитывающих особенности современных серверных архитектур. Рассмотрим некоторые из них.

Наиболее простой организацией колоночного хранилища является добавление к каждой колонке столбца *суррогатных ключей* [47], представляющих собой идентификатор строки в соответствующем отношении. В этом случае мы получаем так называемое *двухстолбцовое представление* с суррогатным ключом (рис. 2 (b)). В работе [73] предложен механизм *виртуальных ключей*. В этом случае роль суррогатного ключа играет порядковый номер значения в столбце. Учитывая тот факт, что все значения столбца имеют одинаковый тип (занимают одинаковое количество байт) и хранятся в непрерывной области памяти, по порядковому номеру значения можно определить область памяти, занимаемой этим значением. Реконструкция *i*-того кортежа отношения осуществляется путем выборки и соединения *i*-тых значений столбцов, принадлежащих этому отношению. В этом случае мы приходим к одностолбцовому представлению (рис. 2 (a)), что позволяет экономить память.

В работах [26, 74] предлагается подход, получивший название «*отложенная материализация*» (*late materialization*). Этот подход предполагает как можно более позднюю реконструкцию кортежей результирующего отношения. Для некоторых классов запросов колоночная СУБД вообще может избежать соединения колонок в кортежи. В этом случае отложенная материализация подразумевает, что колоночная СУБД не только хранит данные в колоночном виде, но и обрабатывает их в колоночном формате. Это позволяет существенно увеличить производительность за счет более эффективного использования иерархической памяти.

Колоночное представление данных является более адаптированным для кэш-памяти процессора, чем строковое. Колоночное хранение позволяет выбирать из оперативной памяти в кэш последовательно значения одного атрибута таблицы. В результате кэш-строки не «засоряются» значениями других атрибутов таблицы, что уменьшает количество кэш-промахов особенно для аналитических запросов [36]. Использование векторизованных команд позволяет еще более повысить скорость выполнения запросов в колоночных

хранилищах [102, 108]. Еще одно преимущество колоночного представления заключается в возможности эффективного применения механизма предварительной выборки данных [132]. Этот механизм предполагает предварительную выборку данных в кэш, что ускоряет к ним доступ.

В работах [143, 25] исследуется эффективность сжатия данных в колоночных хранилищах. Поскольку в колонке все значения имеют одинаковый тип, то для каждой колонки может быть подобран наиболее эффективный метод сжатия, специфичный для данного типа. Максимальный эффект от сжатия достигается на отсортированных данных, когда столбец содержит большие группы повторяющихся значений. Для повышения скорости выполнения запросов наиболее подходит «легковесное» сжатие данных (lightweight data compression) [143], при котором нагрузка на процессор для сжатия/распаковки данных не перевешивает выгоду от уменьшения времени на передачу сжатых данных [46, 136]. В [25, 27, 87] показано, что выполнение операций над данными в сжатом формате позволяет на порядок повысить производительность обработки запросов.

1.4. Обзор работ по теме диссертации

В данном разделе делается обзор работ, наиболее близко относящихся к теме диссертации.

Одним из главных преимуществом строчных хранилищ является наличие в строковых СУБД мощных процедур оптимизации запросов, разработанных на базе реляционной модели. Строковые СУБД также имеют большое преимущество в скорости обработки запросов класса OLTP. В соответствии с этим в исследовательском сообществе баз данных были предприняты интенсивные усилия по интеграции преимуществ столбцовой модели хранения данных в строковые СУБД [24].

В работе [109] предложена новая схема зеркалирования данных, получившая название «*разбитое зеркало*» (*fractured mirror*). Этот подход предполагает гибридную схему хранения данных, включающую в себя и строковое и колоночное представления. На базе строкового представления выполняются операции модификации базы данных, а на базе колоночного – операции чтения и анализа данных. Изменения, производимые в строковом представлении, переносятся в колоночное представление с помощью фонового процесса. Все таблицы в строчном представлении делятся на фрагменты, распределяемые по различным дискам. Каждому строчному фрагменту на этом же диске сопоставляется его зеркальная копия в столбцовой представлении с использованием виртуальных ключей. Указанная схема хорошо подходит для параллельного выполнения запросов, не требующего обменов данными между узлами. Однако, при выполнении сложных запросов, требующих массовых обменов данными между процессорными узлами, данная схема приводит к большим накладным расходам на передачу сообщений. Предложенный подход предполагает для каждого запроса создание гибридного плана, состоящего фактически из комбинации двух планов: один – для строчного представления и второй – для колоночно представления. Оптимизация таких гибридных запросов порождает большое количество трудно решаемых проблем.

В работах [47, 80] была предложена модель хранения данных *DSM* (*Decomposition Storage Model*), которая предполагает декластеризацию каждого отношения на столбцы с использованием суррогатных ключей (см. рис. 2 (b) на стр. 26). В работе [24] была предпринята попытка эмулировать DSM в рамках реляционной СУБД System X. Каждое отношение разбивалось на колонки по числу атрибутов исходного отношения. Каждая колонка представлялась в виде бинарной (двухстолбцовой) таблицы. Первый столбец содержал суррогатный ключ, идентифицирующий соответствующую строку в

исходной таблице, второй столбец содержал значение атрибута. При выполнении запроса с диска загружались только те столбцы, которые соответствуют атрибутам, указанным в запросе. Происходило их соединение по суррогатному ключу в «урезанные» таблицы, над которыми выполнялся запрос. В System X для этих целей по умолчанию используется хеш-соединение, которое оказалось очень дорогостоящим. Для исправления этого недостатка была предпринята попытка вычислять соединения с помощью кластеризованных индексов, создаваемых для каждого столбца, однако из-за обращения к индексам накладные расходы оказались еще выше.

У модели DSM имеются две проблемы. Во-первых, требуется хранение в каждой бинарной таблице столбца с суррогатными ключами, что приводит к излишним затратам дисковой памяти и дополнительным обменам с диском. Во-вторых, в большинстве строчных хранилищ вместе с каждым кортежем хранится относительно крупный заголовок, что также вызывает излишние расходы дисковой памяти (в колоночных хранилищах заголовки хранятся в отдельных столбцах во избежание этих накладных расходов). Для преодоления проблем этих проблем в [24] был исследован подход, при котором отношения хранятся в обычном строчном виде, но на каждом столбце каждой таблицы определяется индекс в виде B-дерева. При выполнении SQL-запросов генерировались *планы выполнения запросов, использующие только индексы* (*index-only plan*). При выполнении таких планов никогда не производится доступ к реальным кортежам на диске. Хотя индексы явно хранят идентификаторы хранимых записей, каждое значение соответствующего столбца сохраняется только один раз, и обычно доступ к значениям столбцов через индексы сопровождается меньшими накладными расходами, поскольку в индексе не хранятся заголовки кортежей. Проведенные эксперименты показали, что такой подход демонстрирует существенно худшую производительность по сравнению с колоночными СУБД.

Еще один подход, рассмотренный в [24], использует *материализованные представления*. При применении этого подхода для каждого класса запросов создается оптимальный набор материализованных представлений, в котором содержатся только столбцы, требуемые в запросах этого класса. В этих представлениях не выполняется соединение столбцов разных таблиц. Цель этой стратегии состоит в том, чтобы дать возможность СУБД производить доступ только к тем данным на диске, которые действительно требуются. Этот подход работает лучше, чем подходы, базирующиеся на использовании DSM и index-only plan. Однако его применения требуется предварительного знания рабочей нагрузки, что существенно ограничивает его использование на практике.

В [43] предпринимается еще одна попытка совместить преимущества строчного и колоночного хранилищ в рамках реляционной СУБД. Предлагается хранить данные в сжатой форме, используя отдельную таблицу (названную *c-таблицей*) для каждой колонки в исходной реляционной схеме (аналогично методу вертикальной декомпозиции отношений). Для сжатия данных используется *метод кодирования по длинам периодов* (*RLE — Run-Length Encoding*) [25]. На запросах класса OLTP данный метод показал производительность, сравнимую с колоночными СУБД. Однако, данная схема предполагает создание с-таблицы для каждого атрибута, и, кроме этого, создание большого количества индексов для каждой с-таблицы, что приводит к большим затратам памяти, отводимой под хранилище. В дополнение к сказанному, зависимости между кортежами в с-таблицах приводят к неоправданно высокой стоимости операции добавления новых записей, даже для приложений с нечастыми обновлениями.

В [52] описан еще один подход к внедрению в строковую СУБД колоночных методов обработки запросов. В основе этого метода лежат планы вы-

полнения запросов, использующие только индексы и специальные операторы Index Merge, Index Merge Join, Index Hash Join, встраиваемые в ядро СУБД. Предлагаемые методы ориентированы на использование твердотельных накопителей (SSD) и многоядерных процессоров. Модифицированная таким образом строковая СУБД способна показать на OLAP-запросах производительность сравнимую и даже превосходящую производительность колоночных СУБД.

В работе [84] описываются новые вспомогательные структуры данных – *индексы колоночной памяти* (*column store indexes*), внедренные в Microsoft SQL Server 11. Индексы колоночной памяти представляют собой колоночное хранилище в чистом виде, так как данные различных колонок хранятся в отдельных дисковых страницах, что позволяет значительно увеличить производительность операций ввода-вывода. Для повышения производительности при выполнении запросов класса OLTP пользователь должен создать индексы колоночной памяти для соответствующих таблиц. Решение об использовании индексов колоночной памяти в том или ином случае принимает СУБД, как это имеет место и для обычных индексов в виде B-деревьев. Авторы иллюстрируют преимущества новых индексов на тесте TPC-DS [129]. Для таблицы catalog_sales (каталог продаж) создается индекс колоночной памяти, содержащий все 34 колонки этой таблицы. Таким образом, наряду со строчным хранилищем в Microsoft SQL Server 11 создается колоночное хранилище, в котором полностью дублируется информация для определенных колонок определенных таблиц. Построение индекса колоночной памяти происходит следующим образом. Исходная таблица делится на последовательные группы строк одинаковой длины. Столбцы значений атрибутов каждой группы кодируются и сжимаются независимо друг от друга. В результате получаются сжатые колоночные сегменты, каждый из которых

сохраняется в виде самостоятельного BLOB (Binary Large Object) [118]. На этапе кодирования данные преобразуются в целые числа. Поддерживается две стратегии: 1) кодирование значения и 2) кодирование путем перечисления (подобно перечислимому типу в универсальных языках программирования). Затем осуществляется сжатие колонки методом RLE. Для достижения максимального от сжатия, исходные группы строк сортируются с помощью специального алгоритма Vertipaq. Для обработки запросов к колоночному хранилищу в Microsoft SQL Server 11 реализованы специальные операторы, поддерживающие *блочную обработку данных* [107] (в исполнителе для строчного хранилища используется традиционная покортежная обработка). Следует отметить, что блочные операторы не применимы к данным, хранящимся в сточной форме. Таким образом, в Microsoft SQL Server 11 фактически реализованы два независимых исполнителя запросов: один для строчного хранилища, другой для колоночного. СУБД при анализе SQL-запроса решает какому из двух исполнителей адресовать его выполнение. СУБД также обеспечивает механизмы синхронизации данных в обоих хранилищах.

Анализ рассмотренных решений показывает [24], что нельзя получить выгоду от хранения данных по столбцам, воспользовавшись системой баз данных со строковым хранением с вертикально разделенной схемой, либо проиндексировав все столбцы, чтобы обеспечить к ним независимый доступ. Системы баз данных с построчным хранением обладают существенно меньшей производительностью по сравнению с системами баз данных с поколоночным хранением на эталонном тестовом наборе для хранилищ данных Star Schema Benchmark (SSBM), [104, 105, 106]. Разница в производительности демонстрирует, что между системами имеются существенные различия на уровне выполнения запросов (кроме очевидных различий на уровне хранения).

Рис. 3. Пространство проектных решений при разработке СУБД для больших данных.

1.5. Выводы по главе 1

Анализ современных тенденций в развитии аппаратного обеспечения и технологий баз данных, проведенный в первой главе, говорит в пользу оптимальности следующего выбора среди возможных решений при разработке СУБД для обработки больших данных, схематично изображенного на рис. 3. Перспективным решением является создание колоночного сопроцессора КСОП (*CCOP – Columnar COProcessor*), совместимого с реляционной СУБД. Колоночный сопроцессор должен поддерживать распределенные *колоночные индексы*, постоянно хранимые в оперативной памяти кластерной вычислительной системы с многоядерными процессорными устройствами. Для взаимодействия с КСОП СУБД должна эмулировать материализационную модель обработки запроса. Суть материализационной модели состоит в том, что промежуточные отношения вычисляются полностью, за исключением атрибутов, не входящих в предикаты вышестоящих реляционных операций. Это достигается переписыванием исходного SQL-запроса в последовательность запросов, использующих материализуемые представления. При таком

подходе любая операция в плане выполнения запроса может быть заменена на вызов КСОП при условии, что в его памяти существуют необходимые колоночные индексы. В заключение отметим, что для сокращения расходов на разработку и сопровождение КСОП, он должен использовать аппаратно-независимые алгоритмы. Временные потери, связанные с отсутствием тонкого тюнинга под конкретную аппаратную платформу, должны компенсироваться хорошей масштабируемостью всех основных алгоритмов КСОП, используемых для выполнения запроса и для модификации колоночных индексов.

ГЛАВА 2. ДОМЕННО-КОЛОНОЧНАЯ МОДЕЛЬ

В данной главе описывается оригинальная формальная доменно-колоночная модель представления данных на базе которой вводится понятие колоночных индексов, доменно-интервальной фрагментации и выполняется декомпозиция реляционных операций.

2.1. Базовые определения и обозначения

Для обозначения реляционных операций будет использоваться нотация из [1]. Под $\pi_{*\setminus A}(R)$ будем понимать проекцию на все атрибуты отношения *R*, за исключением атрибута *A*. С помощью символа «∘» будем обозначать операцию конкатенации двух кортежей:

$$(x_1,\ldots,x_u)\circ(y_1,\ldots,y_v)=(x_1,\ldots,x_u,y_1,\ldots,y_v).$$

Под $R\left(A,B_1,\ldots,B_u\right)$ будем понимать отношение *R* с *суррогатным ключом* A (идентификатором целочисленного типа, однозначно определяющим кортеж) и атрибутами $B_1,\ldots,B_u$, представляющее собой множество кортежей длины $u+1$ вида $(a,b_1,\ldots,b_u)$, где $a\in\mathbb{Z}_{\geq 0}$ и $\forall j\in\{1,\ldots,u\}\left(b_j\in\mathfrak{D}_{B_j}\right)$. Здесь $\mathfrak{D}_{B_j}$ – домен атрибута B_j. Через $r.B_j$ будем обозначать значение атрибута B_j, через $r.A$ – значение суррогатного ключа кортежа *r*: $r=(r.A,r.B_1,\ldots,r.B_u)$. *Суррогатный ключ* отношения *R* обладает свойством $\forall r',r''\in R\left(r'\neq r''\Leftrightarrow r'.A\neq r''.A\right)$. Под *адресом кортежа* r будем понимать значение суррогатного ключа этого кортежа. Для получения кортежа отношения *R* по его адресу будем использовать *функцию разыменования* $\&_R$: $\forall r\in R\left(\&_R(r.A)=r\right)$.

Везде далее будем рассматривать отношения как множества, а не как мультимножества [1]. Это означает, что, если при выполнении некоторой

операции получилось отношение с дубликатами, то к этому отношению по умолчанию применяется операция удаления дубликатов.

2.2. Колоночный индекс

Колоночные индексы позволяют выполнять ресурсоемкие части реляционных операций без обращения к исходным отношениям.

Определение 1. Пусть задано отношение $R(A,B,\ldots)$, $T(R)=n$. Пусть на множестве $\mathfrak{D}_B$ задано отношение линейного порядка. *Колоночным индексом* $I_{R.B}$ атрибута B отношения R будем называть упорядоченное отношение $I_{R.B}(A,B)$, удовлетворяющее следующим свойствам:

$$T(I_{R.B})=n \text{ и } \pi_A(I_{R.B})=\pi_A(R); \tag{1}$$

$$\forall x_1,x_2\in I_{R.B}\left(x_1\le x_2 \Leftrightarrow x_1.B\le x_2.B\right); \tag{2}$$

$$\forall r\in R\left(\forall x\in I_{R.B}\left(r.A=x.A\Rightarrow r.B=x.B\right)\right). \tag{3}$$

Условие (1) означает, что множества значений суррогатных ключей (адресов) индекса и индексируемого отношения совпадают. Условие (2) означает, что элементы индекса упорядочены в порядке возрастания значений атрибута B. Условие (3) означает, что атрибут A элемента индекса содержит адрес кортежа отношения R, имеющего такое же значение атрибута B, как и у данного элемента колоночного индекса.

С содержательной точки зрения колоночный индекс $I_{R.B}$ представляет собой таблицу из двух колонок с именами A и B. Количество строк в колоночном индексе совпадает с количеством строк в индексируемой таблице. Колонка B индекса $I_{R.B}$ включает в себя все значения колонки B таблицы R (с учетом повторяющихся значений), отсортированных в порядке возрастания. Каждая строка x индекса $I_{R.B}$ содержит в колонке A суррогатный

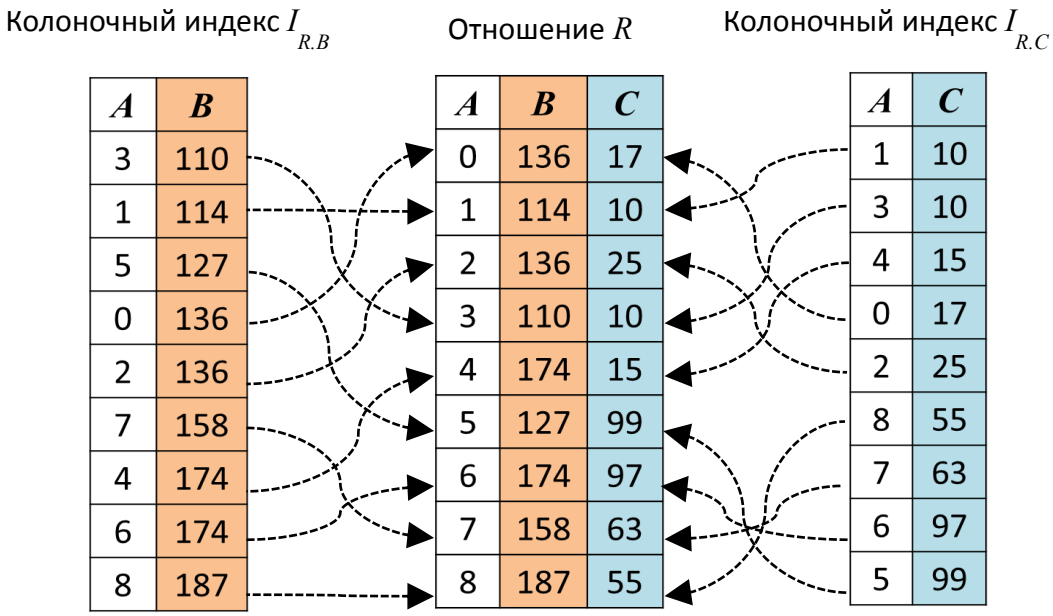


Рис. 4. Колоночные индексы.

ключ (адрес) строки r в таблице R, имеющей такое же значение в колонке B, что и строка x. На рис. 4 представлены примеры двух различных колоночных индексов для одного и того же отношения.

Теорема 1. Пусть задано отношение $R(A,B,\ldots)$. Пусть для отношения R задан колоночный индекс $I_{R.B}$. Тогда

$$\pi_B(I_{R.B}) = \pi_B(R). \tag{4}$$

Другими словами, колоночный индекс $I_{R.B}$ представляет все множество значений атрибута B отношения R с учетом повторяющихся значений.

Доказательство. Возьмем произвольное $b \in \mathfrak{D}_B$. Пусть $T(\sigma_{B=b}(R)) = k$. Без ограничения общности можем считать, что $\forall r \in R(r.A < k \Leftrightarrow r.B = b)$. Тогда из (1) и (3) следует, что $\forall x \in I_{R.B}(x.A < k \Leftrightarrow x.B = b)$. Откуда получаем $T(\sigma_{B=b}(I_{R.B})) = k$. Таким образом (4) имеет место. *Теорема доказана.*

2.3. Доменно-интервальная фрагментация

В данном разделе описывается оригинальный способ фрагментации колоночных индексов, названный доменно-интервальной фрагментацией. Доменно-интервальная фрагментация позволяет осуществлять декомпозицию реляционных операций на основе колоночных индексов таким образом, что ресурсоемкие вычисления над отдельными фрагментами могут выполняться независимо (без обменов данными между процессами).

Определение 2. Пусть на множестве значений домена $\mathfrak{D}_B$ задано отношение линейного порядка. Разобьем множество $\mathfrak{D}_B$ на $k > 0$ непересекающихся интервалов:

$$\left.\begin{array}{l} V_0 = [v_0; v_1), V_1 = [v_1; v_2), \ldots, V_{k-1} = [v_{k-1}; v_k); \\ v_0 < v_1 < \ldots < v_k; \\ \mathfrak{D}_B = \bigcup_{i=0}^{k-1} V_i. \end{array}\right\} \quad (5)$$

Отметим, что в случае $\mathfrak{D}_B = \mathbb{R}$ будем иметь $v_0 = -\infty$ и $v_k = +\infty$.

Функция $\varphi_{\mathfrak{D}_B} : \mathfrak{D}_B \to \{0, \ldots, k-1\}$ называется *доменной функцией фрагментации* для $\mathfrak{D}_B$, если она удовлетворяет следующему условию:

$$\forall i \in \{0, \ldots, k-1\} \left(\forall b \in \mathfrak{D}_B \left(\varphi_{\mathfrak{D}_B}(b) = i \Leftrightarrow b \in V_i \right) \right). \quad (6)$$

Другими словами, доменная функция фрагментации сопоставляет значению b номер интервала, которому это значение принадлежит. Данное определение корректно, так как интервалы $V_0, \ldots, V_{k-1}$ попарно не пересекаются и вместе составляют все множество $\mathfrak{D}_B$.

Определение 3. Пусть задан колоночный индекс $I_{R.B}$ для отношения $R(A, B, \ldots)$ с атрибутом B над доменом $\mathfrak{D}_B$ и доменная функция фрагментации $\varphi_{\mathfrak{D}_B}$. Функция

$$\varphi_{I_{R.B}} : I_{R.B} \to \{0, \ldots, k-1\}, \quad (7)$$

определенная по правилу

$$\forall x \in I_{R.B}\left(\varphi_{I_{R.B}}(x) = \varphi_{\mathfrak{D}_B}(x.B)\right), \tag{8}$$

называется *доменно-интервальной функцией фрагментации* для индекса $I_{R.B}$. Другими словами, функция фрагментации $\varphi_{I_{R.B}}$ сопоставляет каждому кортежу x из $I_{R.B}$ номер доменного интервала, которому принадлежит значение $x.B$.

Определим i-тый фрагмент $\left(i = 0,\ldots,k-1\right)$ индекса $I_{R.B}$ следующим образом:

$$I^i_{R.B} = \left\{x \mid x \in I_{R.B};\ \varphi_{I_{R.B}}(x) = i\right\}. \tag{9}$$

Это означает, что в i-тый фрагмент попадают кортежи, у которых значение атрибута B принадлежит i-тому доменному интервалу. Будем называть фрагментацию, построенную таким образом, *доменно-интервальной*. Количество фрагментов k будем называть *степенью фрагментации*.

Доменно-интервальная фрагментация обладает следующими фундаментальными свойствами, вытекающими непосредственно из ее определения:

$$I_{R.B} = \bigcup_{i=0}^{k-1} I^i_{R.B}; \tag{10}$$

$$\forall i, j \in \{0,\ldots,\mathrm{k}-1\}\left(i \neq j \Rightarrow I^i_{R.B} \cap I^j_{R.B} = \varnothing\right). \tag{11}$$

На рис. 5 схематично изображена фрагментация колоночного индекса, имеющая степень $k = 3$.

Теорема 2. Пусть для колоночного индекса $I_{R.B}$ отношения $R\left(A,B,\ldots\right)$ задана доменно-интервальная фрагментация степени k. Тогда

$$\forall i \in \{0,\ldots,\mathrm{k}-1\}\left(\forall x \in I_{R.B}\left(x \in I^i_{R.B} \Leftrightarrow x.B \in V_i\right)\right). \tag{12}$$

Доказательство. Сначала докажем, что

$$\forall i \in \{0,\ldots,\mathrm{k}-1\}\left(\forall x \in I_{R.B}\left(x \in I^i_{R.B} \Rightarrow x.B \in V_i\right)\right). \tag{13}$$

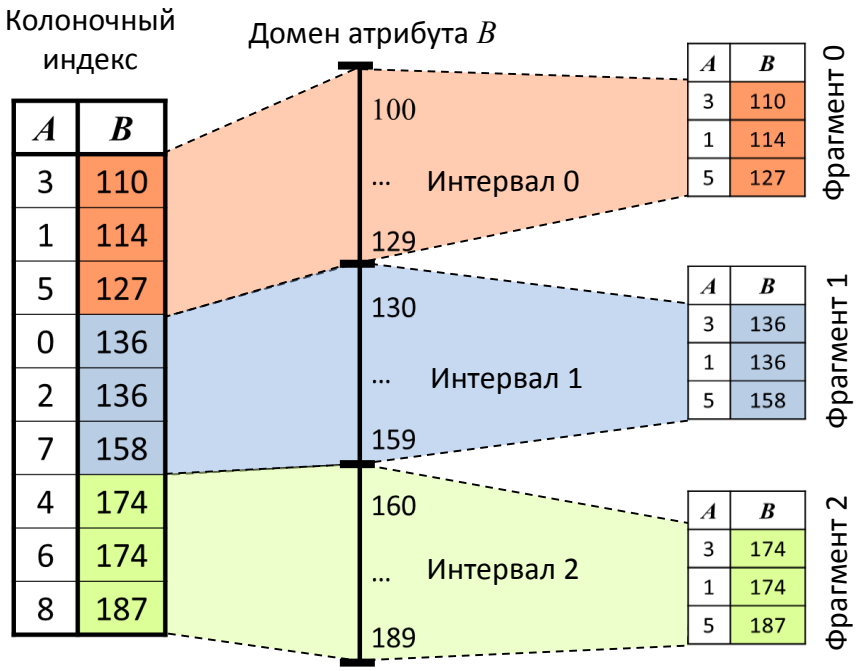


Рис. 5. Фрагментация колоночного индекса.

Пусть $x \in I^i_{R.B}$. Тогда из (9) следует $\varphi_{I_{R.B}}(x) = i$. С учетом (8) получаем $\varphi_{\mathfrak{D}_B}(x.B) = i$. Отсюда и из (6) следует $x.B \in V_i$, то есть (13) имеет место. Теперь докажем, что

$$\forall i \in \{0,\ldots,\mathrm{k}-1\}\left(\forall x \in I_{R.B}\left(x.B \in V_i \Rightarrow x \in I^i_{R.B}\right)\right). \qquad (14)$$

Пусть $x \in I_{R.B}$ и $x.B \in V_i$. Тогда из (6) следует, что $\varphi_{\mathfrak{D}_B}(x.B) = i$. С учетом (8) получаем $\varphi_{\mathfrak{D}_B}(x.B) = \varphi_{I_{R.B}}(x) = i$. Отсюда и из (9) следует, что $x \in I^i_{R.B}$, то есть (14) имеет место. *Теорема доказана.*

2.4. Транзитивная фрагментация

В данном разделе описывается транзитивная фрагментация одного колоночного индекса относительно другого для атрибутов, принадлежащих одному и тому же отношению. Транзитивная фрагментация будет использована при декомпозиции операций проекции (раздел 2.5.1), выбора (раздел 2.5.2), удаления дубликатов (раздел 2.5.3) и группировки (раздел 2.5.4).

Определение 4. Пусть для отношения $R(A,B,C,\ldots)$ заданы колоночные индексы $I_{R.B}$ и $I_{R.C}$. *Транзитивной фрагментацией* индекса $I_{R.C}$ относительно индекса $I_{R.B}$ называется фрагментация, задаваемая функцией $\ddot{\varphi}_{I_{R.C}} : I_{R.C} \to \{0,\ldots,k-1\}$, удовлетворяющей условию $\forall x \in I_{R.C}$:

$$\ddot{\varphi}_{I_{R.C}}(x) = \varphi_{I_{R.B}}\left(\sigma_{A=x.A}(I_{R.B})\right). \quad (15)$$

Транзитивная фрагментация позволяет разместить на одном и том же узле элементы колоночных индексов, соответствующие одному кортежу индексируемого отношения. Пример использования транзитивной фрагментации изображен на рис. 6.

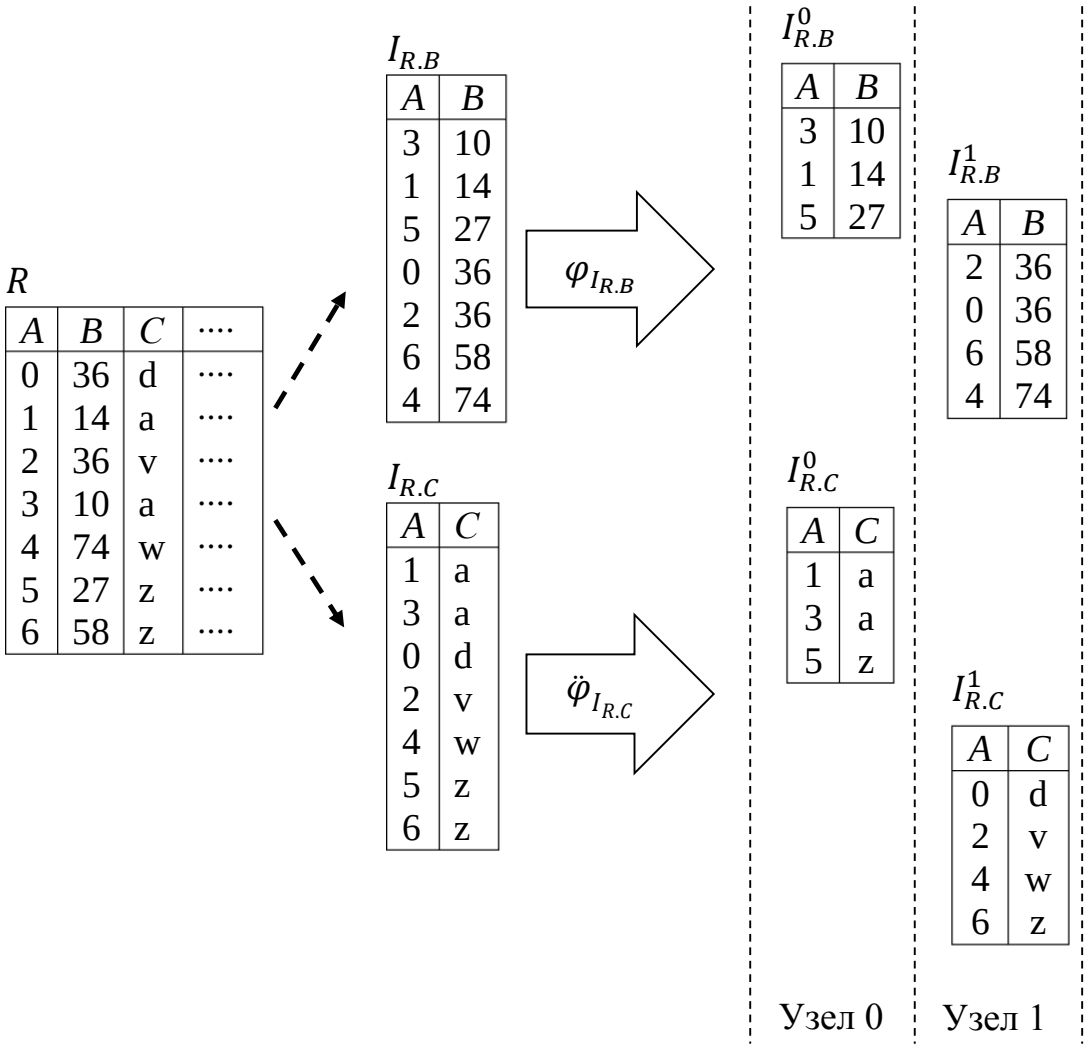


Рис. 6. Транзитивная фрагментация.

Здесь для отношения R строятся два колоночных индекса $I_{R.B}$ и $I_{R.C}$. В качестве функции фрагментации для колоночного индекса $I_{R.B}$ используется функция

$$\varphi_{I_{R.B}}(x) = \begin{cases} 0, \ при \ x.B < 30 \\ 1, \ при \ x.B \geq 30 \end{cases}.$$

Для транзитивной фрагментации колоночного индекса $I_{R.C}$ используется функция $\ddot{\varphi}_{I_{R.C}}$, определяемая формулой (15).

2.5. Декомпозиция реляционных операций с использованием фрагментированных колоночных индексов

В данном разделе рассматривается декомпозиция реляционных операций на основе использования фрагментированных колоночных индексов. Декомпозиция заключается в разбиении ресурсоемких вычислений на отдельные подзадачи, которые могут выполняться в виде независимых процессов, не требующих обменов данными.

2.5.1. Проекция

В данном разделе рассматривается декомпозиция операции проекции вида $\pi_{B,C_1,\ldots,C_u}(R)$. Алгоритм выполнения проекции в строчных СУБД не содержит ресурсоемких вычислений, однако он требует чтения с диска всего отношения. В отличие от этого, предлагаемый ниже алгоритм использует только те столбцы (колоночные индексы) отношения, которые вовлекаются в проекцию. Кроме этого, он вообще не предполагает чтений с диска, так как колоночные индексы хранятся в оперативной памяти.

Пусть задано отношение $R(A,B,C_1,\ldots,C_u,\ldots)$ с суррогатным ключом A. Пусть имеется колоночный индекс $I_{R.B}(A,B)$. Пусть также имеются колоночные индексы $I_{R.C_1}(A,C_1),\ldots,I_{R.C_u}(A,C_u)$. Пусть для индекса $I_{R.B}$ задана доменно-интервальная фрагментация степени k:

$$I_{R.B}=\bigcup_{i=0}^{k-1}I_{R.B}^{i}. \tag{16}$$

Пусть для индексов $I_{R.C_1},\ldots,I_{R.C_u}$ задана транзитивная относительно $I_{R.B}$ фрагментация:

$$\forall j\in\{1,\ldots,u\}\left(I_{R.C_j}=\bigcup_{i=0}^{k-1}I_{R.C_j}^{i}\right). \tag{17}$$

Положим

$$P_i=I_{R.B}^{i}\bowtie I_{R.C_1}^{i}\bowtie\ldots\bowtie I_{R.C_u}^{i} \tag{18}$$

для всех $i=0,\ldots,k-1$. Определим

$$P=\bigcup_{i=0}^{k-1}P_i\ . \tag{19}$$

Построим отношение $Q(B,C_1,\ldots,C_u)$ следующим образом:

$$Q=\pi_{B,C_1,\ldots,C_u}(P). \tag{20}$$

Теорема 3. $Q=\pi_{B,C_1,\ldots,C_u}(R)$.

Доказательство. Сначала докажем, что

$$Q\subset\pi_{B,C_1,\ldots,C_u}(R). \tag{21}$$

Пусть $(b,c_1,\ldots,c_u)\in Q$. В силу (20) и (19) существуют a и i такие, что $(a,b,c_1,\ldots,c_u)\in P_i$. В силу (18) имеем:

$$(a,b)\in I_{R.B};$$
$$(a,c_1)\in I_{R.C_1};$$
$$\ldots\ldots\ldots$$
$$(a,c_u)\in I_{R.C_u}.$$

По определению колоночного индекса (свойства (1) и (3)) отсюда следует, что существует $r \in R$ такой, что $r.A = a; r.B = b; r.C_1 = c_1; \ldots; r.C_u = c_u$, откуда следует $(b, c_1, \ldots, c_u) \in \pi_{B,C_1,\ldots,C_u}(R)$, то есть (21) имеет место.

Теперь покажем, что

$$Q \supset \pi_{B,C_1,\ldots,C_u}(R). \tag{22}$$

Пусть $(b, c_1, \ldots, c_u) \in \pi_{B,C_1,\ldots,C_u}(R)$. Тогда существует $r \in R$ такой, что $r.B = b;\ r.C_1 = c_1;\ \ldots\ ;\ r.C_u = c_u$. Положим $a = r.A$. По определению колоночного индекса (свойства (1) и (3)) отсюда следует что

$$\begin{aligned} &(a, b) \in I_{R.B}; \\ &(a, c_1) \in I_{R.C_1}; \\ &\ldots\ldots\ldots \\ &(a, c_u) \in I_{R.C_u}. \end{aligned} \tag{23}$$

В силу свойства (10) доменно-интервальной фрагментации существует i такой, что $(a, b) \in I^i_{R.B}$. По определению транзитивной относительно $I_{R.B}$ фрагментации отсюда, с учетом (23), следует

$$\begin{aligned} &(a, c_1) \in I^i_{R.C_1}; \\ &\ldots\ldots\ldots \\ &(a, c_u) \in I^i_{R.C_u}. \end{aligned}$$

Тогда, в силу (18), $(a, b, c_1, \ldots, c_u) \in P_i$. Учитывая (19), получаем $(a, b, c_1, \ldots, c_u) \in P$, то есть $(b, c_1, \ldots, c_u) \in \pi_{B,C_1,\ldots,C_u}(P)$. Принимая во внимание (20), имеем $(b, c_1, \ldots, c_u) \in Q$, что означает, что (22) имеет место. *Теорема доказана*.

2.5.2. Выбор

В данном разделе рассматривается декомпозиция операции выбора вида $\sigma_\theta(R)$, где θ – некоторое условие, накладываемое на атрибуты отношения R.

Пусть имеется отношение $R(A,B,C_1,\ldots,C_u,D_1,\ldots,D_w)$ с суррогатным ключом A. Рассмотрим операцию выбора вида $\sigma_{\theta(B,C_1,\ldots,C_u)}(R)$, где $\theta(B,C_1,\ldots,C_u)$ – некоторое условие, зависящее от значений атрибутов $B,C_1,\ldots,C_u$ отношения R. Пусть имеется колоночный индекс $I_{R.B}$. Пусть также имеются колоночные индексы:

$$I_{R.C_1},\ldots,I_{R.C_u}.$$

Пусть для индекса $I_{R.B}$ задана доменно-интервальная фрагментация степени k:

$$I_{R.B}=\bigcup_{i=0}^{k-1}I_{R.B}^i. \tag{24}$$

Пусть для индексов $I_{R.C_1},\ldots,I_{R.C_u}$ и $I_{R.D_1},\ldots,I_{R.D_w}$ задана транзитивная относительно $I_{R.B}$ фрагментация:

$$\forall j\in\{1,\ldots,u\}\left(I_{R.C_j}=\bigcup_{i=0}^{k-1}I_{R.C_j}^i\right). \tag{25}$$

Положим

$$P_i=\pi_A\left(\sigma_{\theta(B,C_1,\ldots,C_u)}\left(I_{R.B}^i\bowtie I_{R.C_1}^i\bowtie\ldots\bowtie I_{R.C_u}^i\right)\right) \tag{26}$$

для всех $i=0,\ldots,k-1$. Определим

$$P=\bigcup_{i=0}^{k-1}P_i\ . \tag{27}$$

Построим отношение $Q(A,B,C_1,\ldots,C_u,D_1,\ldots,D_w)$ следующим образом:

$$Q=\{\&_R(p.A)\mid p\in P\}. \tag{28}$$

Теорема 4. $Q=\sigma_{\theta(B,C_1,\ldots,C_u)}(R)$.

Доказательство. Сначала докажем, что

$$Q\subset\sigma_{\theta(B,C_1,\ldots,C_u)}(R). \tag{29}$$

Пусть $(a,b,c_1,\ldots,c_u,d_1,\ldots,d_w)\in Q$. Тогда из (28) и (1) следует, что существует $r\in R$ такой, что

$$r.A = a; r.B = b; r.C_1 = c_1; \ldots; r.C_u = c_u; r.D_1 = d_1; \ldots; r.D_w = d_w. \quad (30)$$

С другой стороны, в силу (28) существует $p \in P$ такой, что $(a) \in P$. Тогда в силу (27) существует i такой, что $(a) \in P_i$. Отсюда, с учетом (26), существуют $b', c'_1, \ldots, c'_u$ такие, что

$$(a, b', c'_1, \ldots, c'_u) \in \sigma_{\theta(B,C_1,\ldots,C_u)} \left(I^i_{R.B} \bowtie I^i_{R.C_1} \bowtie \ldots \bowtie I^i_{R.C_u} \right). \quad (31)$$

Отсюда следует, что

$$\theta(b', c'_1, \ldots, c'_u) = true \quad (32)$$

и

$$(a, b') \in I_{R.B};$$
$$(a, c'_1) \in I_{R.C_1};$$
$$\ldots\ldots\ldots$$
$$(a, c'_u) \in I_{R.C_u}.$$

По определению колоночного индекса (свойства (1) и (3)) отсюда следует, что существует $r' \in R$ такой, что $r'.A = a$, $r'.B = b'$, $r'.C_1 = c'_1, \ldots, r'.C_u = c'_u$. Принимая во внимание (32), тогда имеем $\theta(r'.B, r'.C_1, \ldots, r'.C_u) = true$, то есть $r' \in \sigma_{\theta(B,C_1,\ldots,C_u)}(R)$. Поскольку $r'.A = a = r.A$ и A – суррогатный ключ в R, с учетом (30), отсюда получаем $(a, b, c_1, \ldots, c_u, d_1, \ldots, d_w) \in \sigma_{\theta(B,C_1,\ldots,C_u)}(R)$, то есть (29) имеет место.

Теперь покажем, что

$$Q \supset \sigma_{\theta(B,C_1,\ldots,C_u)}(R). \quad (33)$$

Пусть $(a, b, c_1, \ldots, c_u, d_1, \ldots, d_w) \in \sigma_{\theta(B,C_1,\ldots,C_u)}(R)$. Тогда существует $r \in R$ такой, что

$$r.A = a;\ r.B = b;\ r.C_1 = c_1;\ \ldots;\ r.C_u = c_u;\ r.D_1 = d_1;\ \ldots;\ r.D_w = d_w \quad (34)$$

и

$$\theta(b, c_1, \ldots, c_u) = true. \quad (35)$$

По определению колоночного индекса (свойства (1) и (3)) из (34) следует что

$$\begin{aligned}&(a,b)\in I_{R.B};\\&(a,c_1)\in I_{R.C_1};\\&\ldots\ldots\\&(a,c_u)\in I_{R.C_u}.\end{aligned}\qquad(36)$$

В силу свойства (10) доменно-интервальной фрагментации существует i такой, что $(a,b)\in I^i_{R.B}$. По определению транзитивной относительно $I_{R.B}$ фрагментации отсюда, с учетом (36), следует

$$\begin{aligned}&(a,c_1)\in I^i_{R.C_1};\\&\ldots\ldots\\&(a,c_u)\in I^i_{R.C_u}.\end{aligned}$$

Тогда с учетом (35) из (26) получаем $(a)\in P_i$. Принимая во внимание (27), имеем $(a)\in P$. С учетом (28) и (1) отсюда следует, что существует $r'\in R$ такой, что

$$r'.A=a \qquad(37)$$

и

$$\left(r'.A,r'.B,r'.C_1,\ldots,r'.C_u,r'.D_1,\ldots,r'.D_w\right)\in Q. \qquad(38)$$

Так как A – суррогатный ключ, из (37) следует, что $r'=r$. Вместе с (38) и (34) это дает $(a,b,c_1,\ldots,c_u,d_1,\ldots,d_w)\in Q$, то есть (33) имеет место. *Теорема доказана*.

2.5.3. Удаление дубликатов

Пусть задано отношение $R\left(A,B,C_1,\ldots,C_u\right)$ с суррогатным ключом *A*. Пусть имеется колоночный индекс $I_{R.B}$. Пусть также имеются колоночные индексы $I_{R.C_1},\ldots,I_{R.C_u}$. Пусть для индекса $I_{R.B}$ задана доменно-интервальная фрагментация степени k:

$$I_{R.B}=\bigcup_{i=0}^{k-1}I^i_{R.B}. \qquad(39)$$

Пусть для индексов $I_{R.C_1},\ldots,I_{R.C_u}$ задана транзитивная относительно $I_{R.B}$ фрагментация:

$$\forall j \in \{1,\ldots,u\}\left(I_{R.C_j} = \bigcup_{i=0}^{k-1} I^i_{R.C_j} \right). \tag{40}$$

Положим

$$P_i = \pi_A\left(\gamma_{\min(A)\to A,B,C_1,\ldots,C_u}\left(I^i_{R.B} \bowtie I^i_{R.C_1} \bowtie \ldots \bowtie I^i_{R.C_u}\right)\right) \tag{41}$$

для всех $i = 0,\ldots,k-1$. Определим

$$P = \bigcup_{i=0}^{k-1} P_i \ . \tag{42}$$

Построим отношение $Q(B,C_1,\ldots,C_u)$ следующим образом:

$$Q = \left\{\left(\&_R(p.A).B, \&_R(p.A).C_1,\ldots,\&_R(p.A).C_u\right) \mid p \in P\right\}. \tag{43}$$

Теорема 5. $Q = \delta\left(\pi_{B,C_1,\ldots,C_u}(R)\right)$.

Доказательство. Сначала докажем, что

$$Q \subset \delta\left(\pi_{B,C_1,\ldots,C_u}(R)\right). \tag{44}$$

Пусть $(b,c_1,\ldots,c_u) \in Q$. Тогда из (43) следует что существует a такой, что $(a) \in P$, и существует $r \in R$ такой, что

$$r.A = a;\ r.B = b;\ r.C_1 = c_1;\ \ldots\ ;\ r.C_u = c_u\,. \tag{45}$$

Следовательно $(b,c_1,\ldots,c_u) \in \delta\left(\pi_{B,C_1,\ldots,C_u}(R)\right)$.

Теперь покажем, что

$$Q \supset \delta\left(\pi_{B,C_1,\ldots,C_u}(R)\right). \tag{46}$$

Пусть

$$(b,c_1,\ldots,c_u) \in \delta\left(\pi_{B,C_1,\ldots,C_u}(R)\right).$$

Положим

$$R'(A,B,C_1,\ldots,C_u) = \left\{r \,\middle|\, r \in R \wedge r.B = b \wedge r.C_1 = c_1 \wedge \ldots \wedge r.C_u = c_u\right\}. \tag{47}$$

Вычислим кортеж

$$r' = \gamma_{\min(A)\to A,B,C_1,\ldots,C_u}\left(R'\right). \quad (48)$$

Очевидно, что

$$r' \in R \quad (49)$$

и

$$r'.B = b;\ r'.C_1 = c_1;\ \ldots;\ r'.C_u = c_u\,. \quad (50)$$

Обозначим

$$a = r'.A\ . \quad (51)$$

По определению колоночного индекса (свойства (1) и (3)) из (49), (50) и (51) следует что

$$(a,b) \in I_{R.B};$$
$$(a,c_1) \in I_{R.C_1};$$
$$\ldots\ldots\ldots$$
$$(a,c_u) \in I_{R.C_u}.$$

В силу свойства (10) доменно-интервальной фрагментации существует i такой, что $(a,b) \in I^i_{R.B}$. По определению транзитивной относительно $I_{R.B}$ фрагментации отсюда следует

$$(a,c_1) \in I^i_{R.C_1};$$
$$\ldots\ldots\ldots$$
$$(a,c_u) \in I^i_{R.C_u}.$$

Тогда с учетом (48) из (41) получаем $(a) \in P_i$. Принимая во внимание (42), имеем $(a) \in P$. С учетом (43) и (1) отсюда следует, что существует $r'' \in R$ такой, что

$$r''.A = a \quad (52)$$

и

$$\left(r''.B, r''.C_1,\ldots,r''.C_u\right) \in Q. \quad (53)$$

Так как A – суррогатный ключ, из (51) и (52) следует, что $r' = r''$. Вместе с (50) и (53) это дает $(b,c_1,\ldots,c_u) \in Q$, то есть (46) имеет место. *Теорема доказана*.

2.5.4. Группировка

В данном разделе рассматривается декомпозиция операции группировки вида $\gamma_{B,C_1,\ldots,C_u,\text{agrf}(D_1,\ldots,D_w)\to F}(R)$.

Пусть задано отношение $R(A,B,C_1,\ldots,C_u,D_1,\ldots,D_w,\ldots)$ с суррогатным ключом *A*. Пусть для атрибутов $D_1,\ldots,D_w$ задана агрегирующая функция **agrf**. Пусть имеется колоночный индекс $I_{R.B}$. Пусть также имеются колоночные индексы:

$$I_{R.C_1},\ldots,I_{R.C_u};$$

$$I_{R.D_1},\ldots,I_{R.D_w}.$$

Пусть для индекса $I_{R.B}$ задана интервальная фрагментация степени k:

$$I_{R.B}=\bigcup_{i=0}^{k-1} I^i_{R.B}. \tag{54}$$

Пусть для индексов $I_{R.C_1},\ldots,I_{R.C_u}$ и $I_{R.D_1},\ldots,I_{R.D_w}$ задана транзитивная относительно $I_{R.B}$ фрагментация:

$$\forall j\in\{1,\ldots,u\}\left(I_{R.C_j}=\bigcup_{i=0}^{k-1} I^i_{R.C_j}\right); \tag{55}$$

$$\forall j\in\{1,\ldots,w\}\left(I_{R.D_j}=\bigcup_{i=0}^{k-1} I^i_{R.D_j}\right). \tag{56}$$

Положим

$$P_i=\pi_{A,F}\left(\gamma_{\min(A)\to A,B,C_1,\ldots,C_u,\text{agrf}(D_1,\ldots,D_w)\to F}\left(I^i_{R.B}\bowtie I^i_{R.C_1}\bowtie\ldots\bowtie I^i_{R.C_u}\bowtie I^i_{R.D_1}\bowtie\ldots\bowtie I^i_{R.D_w}\right)\right) \tag{57}$$

для всех $i=0,\ldots,k-1$. Определим

$$P=\bigcup_{i=0}^{k-1} P_i\ . \tag{58}$$

Построим отношение $Q(B,C_1,\ldots,C_u,F)$ следующим образом:

$$Q=\left\{\left(\&_R(p.A).B,\&_R(p.A).C_1,\ldots,\&_R(p.A).C_u,p.F\right)\mid p\in P\right\}. \tag{59}$$

Теорема 6. $Q=\gamma_{B,C_1,\ldots,C_u,\text{agrf}(D_1,\ldots,D_w)\to F}(R)$.

Доказательство. Сначала докажем, что

$$\pi_{*\setminus F}(Q) = \pi_{*\setminus F}\left(\gamma_{B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(R)\right). \quad (60)$$

Для этого нам достаточно доказать справедливость следующих двух утверждений:

$$\pi_{*\setminus F}(Q) \subset \pi_{*\setminus F}\left(\gamma_{B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(R)\right); \quad (61)$$

и

$$\pi_{*\setminus F}(Q) \supset \pi_{*\setminus F}\left(\gamma_{B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(R)\right). \quad (62)$$

Справедливость утверждения (61) непосредственно следует из (59) и (1).

Покажем справедливость утверждения (62). Пусть

$$(b,c_1,\ldots,c_u) \in \pi_{*\setminus F}\left(\gamma_{B,C_1,\ldots,C_u,\mathrm{agrf}(D_1,\ldots,D_w)\to F}(R)\right).$$

Это означает, что

$$T\left(\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(\gamma_{\min(A)\to A,B,C_1,\ldots,C_u}(R)\right)\right)=1. \quad (63)$$

Положим

$$(a) \in \pi_A\left(\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(\gamma_{\min(A)\to A,B,C_1,\ldots,C_u}(R)\right)\right). \quad (64)$$

С учетом (4) отсюда получаем, что существуют

$$x_B \in I_{R.B},\ x_{C_1} \in I_{R.C_1},\ldots,\ x_{C_u} \in I_{R.C_u}$$

такие, что

$$(x_B.A = a \wedge x_B.B = b) \wedge (x_{C_1}.A = a \wedge x_{C_1}.C_1 = c_1)\cdots\wedge(x_{C_u}.A = a \wedge x_{C_u}.C_u = c_u). \quad (65)$$

Пусть в контексте разбиения (5) $b \in V_l$ $(l \in \{0,\ldots,k-1\})$. Тогда из (12) и из (65) следует, что

$$x_B \in I^l_{R.B}. \quad (66)$$

По определению 4 транзитивной фрагментации из (65) и (66) получаем

$$x_{C_1} \in I^l_{R.C_1},\cdots,x_{C_u} \in I^l_{R.C_u}. \quad (67)$$

Сопоставляя (57), (64), (65) и (67), получаем, что $(a) \in \pi_A(P_l)$. С учетом (58) отсюда следует $(a) \in \pi_A(P)$. Вместе с (59) это дает

$$\left(\&_R(a).B, \&_R(a).C_1, \ldots, \&_R(a).C_u\right) \in \pi_{*\setminus F}\left(Q\right),$$

откуда с учетом (64) получаем $(b, c_1, \ldots, c_u) \in \pi_{*\setminus F}\left(Q\right)$. Таким образом (62), а вместе с ним и (60) имеет место.

Для завершения доказательства теоремы нам достаточно показать, что для любых $q \in Q$ и $g \in \gamma_{B,C_1,\ldots,C_u,\text{agrf}(D_1,\ldots,D_w)\to F}(R)$ справедливо:

$$\left(q.B = g.B \wedge q.C_1 = g.C_1 \wedge \ldots \wedge q.C_1 = g.C_1\right) \Rightarrow \left(q.F = g.F\right). \qquad (68)$$

Докажем (68). Пусть

$$q = (b, c_1, \ldots, c_u, f) \in Q \qquad (69)$$

и

$$g = (b, c_1, \ldots, c_u, f') \in \gamma_{B,C_1,\ldots,C_u,\text{agrf}(D_1,\ldots,D_w)\to F}(R). \qquad (70)$$

Такие q и g существуют в силу (60). Для доказательства (68) достаточно показать, что $f = f'$.

В соответствии с (59) существует

$$p = (a, f) \in P \qquad (71)$$

такой, что

$$(a, b, c_1, \ldots, c_u, \ldots) \in R. \qquad (72)$$

Из (71) и (58) следует, что существует l, такое что

$$p = (a, f) \in P^l. \qquad (73)$$

Отсюда, с учетом (57) получаем

$$(a, b', c_1', \ldots, c_u') \in I^l_{R.B} \bowtie I^l_{R.C_1} \bowtie \ldots \bowtie I^l_{R.C_u}.$$

В силу свойств транзитивной фрагментации и колоночных индексов с учетом (72) отсюда следует, что

$$(a, b, c_1, \ldots, c_u) \in I^l_{R.B} \bowtie I^l_{R.C_1} \bowtie \ldots \bowtie I^l_{R.C_u}. \qquad (74)$$

Вместе с (71) и (57) это дает

$$f = \pi_F\left(\gamma_{B,C_1,\ldots,C_u,\text{agrf}(D_1,\ldots,D_w)\to F}\left(\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(I^l_{R.B} \bowtie I^l_{R.C_1} \bowtie \ldots \bowtie I^l_{R.C_u} \bowtie I^l_{R.D_1} \bowtie \ldots \bowtie I^l_{R.D_w}\right)\right)\right). \qquad (75)$$

С другой стороны, из (70) непосредственно следует, что

$$f' = \pi_F\left(\gamma_{B,C_1,\ldots,C_u,\text{agrf}(D_1,\ldots,D_w)\to F}\left(\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(R\right)\right)\right). \qquad (76)$$

Исходя из формул (75) и (76), условие $f = f'$ равносильно условию

$$\begin{aligned}\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(I^l_{R.B}\bowtie I^l_{R.C_1}\bowtie\ldots\bowtie I^l_{R.C_u}\bowtie I^l_{R.D_1}\bowtie\ldots\bowtie I^l_{R.D_w}\right) = \\ = \sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(\pi_{B,C_1,\ldots,C_u,D_1,\ldots,D_w}\left(R\right)\right).\end{aligned} \qquad (77)$$

Пусть

$$(b,c_1,\ldots,c_u,d_1,\ldots,d_w)\in\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(I^l_{R.B}\bowtie I^l_{R.C_1}\bowtie\ldots\bowtie I^l_{R.C_u}\bowtie I^l_{R.D_1}\bowtie\ldots\bowtie I^l_{R.D_w}\right).$$

Это означает, что существует a' такой, что

$$(a',b)\in I^l_{R.B}\subset I_{R.B};$$

$$(a',b)\in I^l_{R.C_1}\subset I_{R.C_1};$$

$$\ldots$$

$$(a',b)\in I^l_{R.C_u}\subset I_{R.C_u};$$

$$(a',b)\in I^l_{R.D_1}\subset I_{R.D_1};$$

$$\ldots$$

$$(a',b)\in I^l_{R.D_w}\subset I_{R.D_w}.$$

Отсюда в силу свойств колоночного индекса

$$(a',b,c_1,\ldots,c_u,d_1,\ldots,d_w)\in\pi_{A,B,C_1,\ldots,C_u,D_1,\ldots,D_w}\left(R\right),$$

И следовательно

$$(b,c_1,\ldots,c_u,d_1,\ldots,d_w)\in\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(\pi_{B,C_1,\ldots,C_u,D_1,\ldots,D_w}\left(R\right)\right).$$

Пусть теперь

$$(b,c_1,\ldots,c_u,d'_1,\ldots,d'_w)\in\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(\pi_{B,C_1,\ldots,C_u,D_1,\ldots,D_w}\left(R\right)\right). \qquad (78)$$

В силу свойств транзитивной фрагментации и колоночных индексов, принимая во внимание (74), получаем

$$\left(b,c_1,\ldots,c_u,d'_1,\ldots,d'_w\right)\in\left(I^l_{R.B}\bowtie I^l_{R.C_1}\bowtie\ldots\bowtie I^l_{R.C_u}\bowtie I^l_{R.D_1}\bowtie\ldots\bowtie I^l_{R.D_w}\right),$$

откуда немедленно следует

$$\left(b,c_1,\ldots,c_u,d'_1,\ldots,d'_w\right)\in\sigma_{B=b\wedge C_1=c_1\wedge\ldots\wedge C_u=c_u}\left(I^l_{R.B}\bowtie I^l_{R.C_1}\bowtie\ldots\bowtie I^l_{R.C_u}\bowtie I^l_{R.D_1}\bowtie\ldots\bowtie I^l_{R.D_w}\right).$$

Таким образом (77), а значит и (68) имеют место. *Теорема доказана*.

2.5.5. Пересечение

В данном разделе рассматривается декомпозиция операции пересечения вида $\pi_{B_1,\ldots,B_u}(R)\cap\pi_{B_1,\ldots,B_u}(S)$. При этом предполагается, что $\pi_{B_1,\ldots,B_u}(R)$ и $\pi_{B_1,\ldots,B_u}(S)$ не содержат дубликатов.

Пусть заданы два отношения $R\left(A,B_1,\ldots,B_u\right)$ и $S\left(A,B_1,\ldots,B_u\right)$, имеющие одинаковый набор атрибутов. Пусть имеется два набора колоночных индексов по атрибутам $B_1,\ldots,B_u$:

$$I_{R.B_1},\ldots,I_{R.B_u};$$

$$I_{S.B_1},\ldots,I_{S.B_u}.$$

Пусть для всех этих индексов задана доменно-интервальная фрагментация степени k:

$$I_{R.B_j}=\bigcup_{i=0}^{k-1}I^i_{R.B_j};\qquad(79)$$

$$I_{S.B_j}=\bigcup_{i=0}^{k-1}I^i_{S.B_j}.\qquad(80)$$

Положим

$$P^i_j=\pi_{I^i_{R.B_j}.A\to A_R,\ I^i_{S.B_j}.A\to A_S}\left(I^i_{R.B_j}\underset{(I^i_{R.B_j}.B_j=I^i_{S.B_j}.B_j)}{\bowtie}I^i_{S.B_j}\right)\qquad(81)$$

для всех $i=0,\ldots,k-1$ и $j=1,\ldots,u$. Определим

$$P_j=\bigcup_{i=0}^{k-1}P^i_j.\qquad(82)$$

Положим

$$P=\bigcap_{j=1}^{u}P_j.\qquad(83)$$

Построим отношение $Q(A,B_1,\ldots,B_u)$ следующим образом:

$$Q = \left\{ r \mid r \in R \wedge r.A \in \pi_{A_R}(P) \right\}. \tag{84}$$

Теорема 7. $\pi_{B_1,\ldots,B_u}(Q) = \pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S)$.

Доказательство. Сначала докажем, что

$$\pi_{B_1,\ldots,B_u}(Q) \subset \pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S). \tag{85}$$

Пусть

$$(a, b_1, \ldots, b_u) \in Q. \tag{86}$$

Из (84) следует, что

$$(a, b_1, \ldots, b_u) = r \in R \tag{87}$$

и

$$a = r.A \in \pi_{A_R}(P). \tag{88}$$

Отсюда следует, что $\exists p \in P\left(p.A_R = a \wedge p.A_S = a'\right)$. С учетом (83) получаем, что $\forall j \in \{1,\ldots,u\}\left(\exists p \in P_j\left(p.A_R = a \wedge p.A_S = a'\right)\right)$. С учетом (82) отсюда получаем $\forall j \in \{1,\ldots,u\}\left(\exists i \in \{0,\ldots,k-1\}\left(\exists p \in P_j^i\left(p.A_R = a \wedge p.A_S = a'\right)\right)\right)$. Отсюда и из (79)-(81) следует, что

$$\forall j \in \{1,\ldots,u\}\left(\exists x \in I_{R.B_j}\left(\exists y \in I_{S.B_j}\left(x.A = a \wedge x.B_j = y.B_j \wedge y.A = a'\right)\right)\right).$$

По определению колоночного индекса отсюда получаем

$$\forall j \in \{1,\ldots,u\}\left(\exists \tilde{r} \in R\left(\exists \tilde{s} \in S\left(\tilde{r}.A = a \wedge \tilde{r}.B_j = \tilde{s}.B_j \wedge \tilde{s}.A = a'\right)\right)\right).$$

Поскольку A является ключом в R и S, с учетом (87) отсюда следует $(a', b_1, \ldots, b_u) \in S$, то есть $(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S)$. Таким образом (85) имеет место.

Теперь докажем, что

$$\pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S) \subset \pi_{B_1,\ldots,B_u}(Q). \tag{89}$$

Пусть

$$(a, b_1, \ldots, b_u) = r \in R \tag{90}$$

и

$$(a', b_1, \ldots, b_u) = s \in S. \quad (91)$$

Тогда по определению колоночного индекса с учетом (4) имеем

$$\forall j \in \{1, \ldots, u\}\left(\exists x \in I_{R.B_j}\left(\exists y \in I_{S.B_j}\left(x.A = a \wedge x.B_j = b_j = y.B_j \wedge y.A = a'\right)\right)\right).$$

На основе (12) отсюда получаем

$$\forall j \in \{1, \ldots, u\}\left(\exists i \in \{0, \ldots, \mathrm{k}-1\}\left(\exists x \in I^i_{R.B_j}\left(\exists y \in I^i_{S.B_j}\left(x.A = a \wedge x.B_j = y.B_j \wedge y.A = a'\right)\right)\right)\right).$$

С учетом (81) отсюда следует

$$\forall j \in \{1, \ldots, u\}\left(\exists i \in \{0, \ldots, \mathrm{k}-1\}\left(\exists p \in P^i_j\left(p.A_R = a \wedge p.A_S = a'\right)\right)\right).$$

Применяя (82) получаем

$$\forall j \in \{1, \ldots, u\}\left(\exists p \in P_j\left(p.A_R = a \wedge p.A_S = a'\right)\right).$$

Учитывая (83) отсюда имеем $(a, a') \in P$. Вместе с (84) и (90) это дает

$$(a, b_1, \ldots, b_u) \in Q,$$

то есть (89) имеет место. *Теорема доказана*.

2.5.6. Естественное соединение

Пусть заданы два отношения

$$R\left(A, B_1, \ldots, B_u, C_1, \ldots, C_v\right)$$

и

$$S\left(A, B_1, \ldots, B_u, D_1, \ldots, D_w\right).$$

Пусть имеется два набора колоночных индексов по атрибутам $B_1, \ldots, B_u$:

$$I_{R.B_1}, \ldots, I_{R.B_u};$$

$$I_{S.B_1}, \ldots, I_{S.B_u}.$$

Пусть для всех этих индексов задана доменно-интервальная фрагментация степени k:

$$I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j}; \quad (92)$$

$$I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j} . \tag{93}$$

Положим

$$P^i_j = \pi_{I^i_{R.B_j}.A \to A_R,\, I^i_{S.B_j}.A \to A_S} \left(I^i_{R.B_j} \underset{I^i_{R.B_j}.B_j = I^i_{S.B_j}.B_j}{\bowtie} I^i_{S.B_j} \right) \tag{94}$$

для всех $i = 0,\ldots,k-1$ и $j = 1,\ldots,u$. Определим

$$P_j = \bigcup_{i=0}^{k-1} P^i_j . \tag{95}$$

Положим

$$P = \bigcap_{j=1}^{u} P_j . \tag{96}$$

Построим отношение $Q(B_1,\ldots,B_u,C_1,\ldots,C_v,D_1,\ldots,D_w)$ следующим образом:

$$\begin{aligned} Q = \{ & (\&_R(p.A_R).B_1,\ldots,\&_R(p.A_R).B_u, \\ & \&_B(p.A_B).C_1,\ldots,\&_B(p.A_B).C_v, \\ & \&_S(p.A_S).D_1,\ldots,\&_S(p.A_S).D_w) \mid p \in P \}. \end{aligned} \tag{97}$$

Теорема 8. $Q = \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$.

Доказательство. Сначала докажем, что

$$Q \subset \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S). \tag{98}$$

Пусть

$$(b_1,\ldots,b_u,c_1,\ldots,c_v,d_1,\ldots,d_w) \in Q. \tag{99}$$

Из (97) следует, что существуют кортежи *r* и *s* такие, что

$$(a,b_1,\ldots,b_u,c_1,\ldots,c_v) = r \in R, \tag{100}$$

$$(a',b_1',\ldots,b_u',d_1,\ldots,d_w) = s \in S \tag{101}$$

и

$$(r.A, s.A) \in P. \tag{102}$$

Отсюда следует, что $\exists p \in P\left(p.A_R = a \wedge p.A_S = a'\right)$. С учетом (96) получаем, что $\forall j \in \{1,\ldots,u\}\left(\exists p \in P_j\left(p.A_R = a \wedge p.A_S = a'\right)\right)$. С учетом (95) отсюда получаем $\forall j \in \{1,\ldots,u\}\left(\exists i \in \{0,\ldots,k-1\}\left(\exists p \in P_j^i\left(p.A_R = a \wedge p.A_S = a'\right)\right)\right)$. Отсюда и из (92)–(94) следует, что

$$\forall j \in \{1,\ldots,u\}\left(\exists x \in I_{R.B_j}\left(\exists y \in I_{S.B_j}\left(x.A = a \wedge x.B_j = y.B_j \wedge y.A = a'\right)\right)\right).$$

По определению колоночного индекса отсюда получаем

$$\forall j \in \{1,\ldots,u\}\left(\exists \tilde{r} \in R\left(\exists \tilde{s} \in S\left(\tilde{r}.A = a \wedge \tilde{r}.B_j = \tilde{s}.B_j \wedge \tilde{s}.A = a'\right)\right)\right).$$

Поскольку A является ключом в R и S, с учетом (100) и (101) отсюда следует

$$(a', b_1,\ldots,b_u, d_1,\ldots,d_w) \in S,$$

то есть $(b_1,\ldots,b_u,c_1,\ldots,c_v,d_1,\ldots,d_w) \in \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$. Таким образом (98) имеет место.

Теперь докажем, что

$$Q \supset \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S). \tag{103}$$

Пусть $(b_1,\ldots,b_u,c_1,\ldots,c_v,d_1,\ldots,d_w) \in \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$ тогда существуют $r \in R$ и $s \in S$, такие, что

$$r = (a, b_1,\ldots,b_u,c_1,\ldots,c_v) \tag{104}$$

и

$$s = (a', b_1,\ldots,b_u,d_1,\ldots,d_w). \tag{105}$$

Тогда по определению колоночного индекса с учетом (4) имеем

$$\forall j \in \{1,\ldots,u\}\left(\exists x \in I_{R.B_j}\left(\exists y \in I_{S.B_j}\left(x.A = a \wedge x.B_j = b_j = y.B_j \wedge y.A = a'\right)\right)\right).$$

На основе (12) отсюда получаем

$$\forall j \in \{1,\ldots,u\}\left(\exists i \in \{0,\ldots,\mathrm{k}-1\}\left(\exists x \in I^i_{R.B_j}\left(\exists y \in I^i_{S.B_j}\left(x.A = a \wedge x.B_j = y.B_j \wedge y.A = a'\right)\right)\right)\right).$$

С учетом (94) отсюда следует

$$\forall j \in \{1,\ldots,u\}\left(\exists i \in \{0,\ldots,\mathrm{k}-1\}\left(\exists p \in P_j^i\left(p.A_R = a \wedge p.A_S = a'\right)\right)\right).$$

Применяя (95) получаем

$$\forall j \in \{1,\ldots,u\}\left(\exists p \in P_j\left(p.A_R = a \wedge p.A_S = a'\right)\right).$$

Учитывая (96) отсюда имеем $(a,a') \in P$. Вместе с (97), (104) и (105) это дает

$$(b_1,\ldots,b_u,c_1,\ldots,c_v,d_1,\ldots,d_w) \in Q,$$

то есть (103) имеет место. *Теорема доказана*.

2.5.7. Объединение

В данном разделе рассматривается декомпозиция операции объединения двух отношений вида $\pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S)$. При этом предполагается, что $\pi_{B_1,\ldots,B_u}(R)$ и $\pi_{B_1,\ldots,B_u}(S)$ не содержат дубликатов. Результирующее отношение также не должно содержать дубликатов.

Пусть заданы два отношения $R\left(A,B_1,\ldots,B_u\right)$ и $S\left(A,B_1,\ldots,B_u\right)$, имеющие одинаковый набор атрибутов. Пусть имеется два набора колоночных индексов по атрибутам $B_1,\ldots,B_u$:

$$I_{R.B_1},\ldots,I_{R.B_u};$$

$$I_{S.B_1},\ldots,I_{S.B_u}.$$

Пусть для индексов $I_{R.B_1}$ и $I_{S.B_1}$ задана доменно-интервальная фрагментация степени k:

$$I_{R.B_1} = \bigcup_{i=0}^{k-1} I_{R.B_1}^i; \tag{106}$$

$$I_{S.B_1} = \bigcup_{i=0}^{k-1} I_{S.B_1}^i. \tag{107}$$

Пусть для индексов $I_{R.B_2},\ldots,I_{R.B_u}$ и $I_{S.B_2},\ldots,I_{S.B_u}$ задана транзитивная фрагментация относительно $I_{R.B_1}$ и $I_{S.B_1}$ соответственно:

$$\forall j \in \{2,\ldots,u\}\left(I_{R.B_j} = \bigcup_{i=0}^{k-1} I^i_{R.B_j} \right); \tag{108}$$

$$\forall j \in \{2,\ldots,u\}\left(I_{S.B_j} = \bigcup_{i=0}^{k-1} I^i_{S.B_j} \right). \tag{109}$$

Положим для всех $i = 0,\ldots,k-1$ и $j = 1,\ldots,u$

$$\tilde{P}^i_j = \pi_{I^i_{R.B_j}.A \to A_R,\, I^i_{S.B_j}.A \to A_S}\left(I^i_{R.B_j} \underset{(I^i_{R.B_j}.B_j = I^i_{S.B_j}.B_j)}{\bowtie} I^i_{S.B_j} \right). \tag{110}$$

Определим

$$\tilde{P}^i = \bigcap_{j=1}^{u} \tilde{P}^i_j \,. \tag{111}$$

Положим для всех $i = 0,\ldots,k-1$

$$P^i_R = \pi_A\left(I^i_{R.B_1} \right) \tag{112}$$

и

$$P^i_S = \pi_A\left(I^i_{S.B_1} \underset{(I^i_{S.B_1}.A \neq \tilde{P}^i.A_S)}{\bowtie} \tilde{P}^i \right). \tag{113}$$

Определим

$$P_R = \bigcup_{i=0}^{k-1} P^i_R \,, \tag{114}$$

$$P_S = \bigcup_{i=0}^{k-1} P^i_S \,. \tag{115}$$

Построим отношение $Q(A, B_1, \ldots, B_u)$ следующим образом:

$$Q = \left\{ \&_R\left(p.A \right) \mid p \in P_R \right\} \cup \left\{ \&_S\left(p.A \right) \mid p \in P_S \right\}. \tag{116}$$

Теорема 9. $\pi_{B_1,\ldots,B_u}(Q) = \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S)$.

Доказательство. Сначала докажем, что

$$\pi_{B_1,\ldots,B_u}(Q) \subset \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S). \tag{117}$$

Пусть

$$(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(Q). \tag{118}$$

Из (116) следует, что как минимум одно из следующих условий истинно:

$$(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}\left(\left\{\&_R(p.A) \mid p \in P_R\right\}\right); \tag{119}$$

$$(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}\left(\left\{\&_S(p.A) \mid p \in P_S\right\}\right). \tag{120}$$

Предположим сначала, что условие (119) истинно. Тогда существует $p \in P_R$ такой, что

$$\&_R(p.A) = r \in R$$

и

$$r = (a, b_1,\ldots,b_u),$$

где $a = p.A$. Отсюда получаем $(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(R)$. Следовательно $(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S)$, и (117) имеет место. Для случая, когда истинным является условие (120), это утверждение доказывается аналогично (в рассуждениях выше надо просто заменить *R* на *S* и *r* на *s*).

Теперь докажем, что

$$\pi_{B_1,\ldots,B_u}(Q) \supset \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S). \tag{121}$$

По условию, объединение в правой части не должно содержать дубликатов (в предположении, что $\pi_{B_1,\ldots,B_u}(R)$ и $\pi_{B_1,\ldots,B_u}(S)$ не содержат дубликатов). Такое объединение без дубликатов может быть вычислено следующим образом:

$$\tilde{S} = \pi_{S.*}\left(R \underset{R.B_1=S.B_1 \wedge \ldots \wedge R.B_u=S.B_u}{\bowtie} S\right); \tag{122}$$

$$\bar{S} = \pi_{S.*}\left(\tilde{S} \underset{\tilde{S}.A \neq S.A}{\bowtie} S\right); \tag{123}$$

$$\pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S) = \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(\bar{S}). \tag{124}$$

Поэтому условие (121) эквивалентно условию

$$\pi_{B_1,\ldots,B_u}(Q) \supset \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(\bar{S}). \tag{125}$$

Покажем, что последнее истинно. Пусть $(b_1,\ldots,b_u)\in\pi_{B_1,\ldots,B_u}(R)\cup\pi_{B_1,\ldots,B_u}(\bar{S})$. Тогда $(b_1,\ldots,b_u)\in\pi_{B_1,\ldots,B_u}(R)$, либо $(b_1,\ldots,b_u)\in\pi_{B_1,\ldots,B_u}(\bar{S})$. Предположим сначала, что $(b_1,\ldots,b_u)\in\pi_{B_1,\ldots,B_u}(R)$. Это означает, что существует $r\in R$ такой, что

$$r.B_1=b_1,\ldots,r.B_u=b_u. \tag{126}$$

Положим

$$r.A=a. \tag{127}$$

По определению колоночного индекса (свойства (1) и (3)) отсюда следует что $(a,b_1)\in I_{R.B_1}$. В силу свойства (10) доменно-интервальной фрагментации существует i такой, что $(a,b_1)\in I^i_{R.B_1}$. Тогда с учетом (112) и (114) получаем

$$p=(a)\in P_R. \tag{128}$$

Положим

$$r'=\&_R\left(p.A\right). \tag{129}$$

По построению

$$r'.A=a. \tag{130}$$

Принимая во внимание, что A – суррогатный ключ, из (130) и (127) следует

$$r=r'. \tag{131}$$

Из (116), (128), (129), (131) следует, что $r\in Q$. Вместе с (126) это дает $(b_1,\ldots,b_u)\in\pi_{B_1,\ldots,B_u}(Q)$, то есть (121) имеет место для рассмотренного случая.

Предположим теперь, что

$$(b_1,\ldots,b_u)\in\pi_{B_1,\ldots,B_u}(\bar{S}). \tag{132}$$

Это означает, что существует $\bar{s}\in\bar{S}$ такой, что

$$\bar{s}.B_1=b_1,\ldots,\bar{s}.B_u=b_u. \tag{133}$$

Положим

$$\bar{s}.A=a. \tag{134}$$

Из (123) следует, что $\overline{s} \in S$. По определению колоночного индекса из (133) следует что

$$\begin{array}{l} (a,b_1) \in I_{S.B_1}; \\ \ldots\ldots\ldots \\ (a,b_u) \in I_{S.B_u}. \end{array} \qquad (135)$$

В силу свойства (10) доменно-интервальной фрагментации существует i такой, что $(a,b_1) \in I^i_{S.B}$. По определению транзитивной относительно $I_{S.B_1}$ фрагментации отсюда, с учетом (135), имеем

Покажем, что

$$a \notin \tilde{P}^i \qquad (136)$$

$$(a,b_2) \in I^i_{S.B_2};$$

Предположим прот…… ив $a \in \tilde{P}^i$ ное, то есть . Тогда существует

$$(a,b_u) \in I^i_{S.B_u}.$$

$(a',b'_1,\ldots,b'_u) \in R$ такой, что $(a',a) \in \tilde{P}^i$. По определению колоночного индекса $(a',b'_1) \in I_{R.B_1}$. В силу свойств доменно-интервальной фрагментации $(a',b'_1) \in I^i_{R.B_1}$. По определению транзитивной относительно $I_{R.B_1}$ фрагментации отсюда следует

$$\begin{array}{l} (a',b'_2) \in I^i_{R.B_2}; \\ \ldots\ldots\ldots \\ (a',b'_u) \in I^i_{R.B_u}. \end{array}$$

Но тогда с учетом (111) и (110) получаем

$$\begin{array}{l} b'_1 = b_1; \\ \ldots \\ b'_u = b_u. \end{array}$$

Отсюда в силу (122) следует $(a,b_1,\ldots,b_u)\in\tilde{S}$. Принимая во внимание (123), получаем $(a,b_1,\ldots,b_u)\notin\overline{S}$. Так как мы предположили, что $\pi_{B_1,\ldots,B_u}(S)$ не содержит дубликатов, отсюда следует $(b_1,\ldots,b_u)\notin\pi_{B_1,\ldots,B_u}(\overline{S})$. Получили противоречие с (132). Таким образом, (136) истинно. Тогда из (113) следует, что $(a)\in P_S^i$. С учетом (115) получаем $(a)\in P_S$. Теперь из (116), (134), (133) и (123) следует, что $(a,b_1,\ldots,b_u)\in Q$, то есть $(b_1,\ldots,b_u)\in\pi_{B_1,\ldots,B_u}(Q)$. *Теорема доказана*.

2.6. Колоночный хеш-индекс

Колоночный хеш-индекс позволяет использовать один колоночный индекс для индексирования нескольких атрибутов одного отношения.

Определение 5. Пусть задано отношение $R(A,B_1,\ldots,B_u,\ldots)$. Пусть задана инъективная хеш-функция $h:\mathfrak{D}_{B_1}\times\ldots\times\mathfrak{D}_{B_u}\to\mathbb{Z}_{\geq 0}$. *Колоночным хеш-индексом* $I_h(A,H)$ атрибутов $B_1,\ldots,B_u$ отношения R будем называть упорядоченное отношение, удовлетворяющее тождеству:

$$I_h=\tau_H\left(\pi_{A,h(B_1,\ldots,B_u)\to H}(R)\right). \qquad (137)$$

Колоночный хеш-индекс обладает следующим основным свойством:

$$\forall r',r''\in R\left(r'.B_1=r''.B_1\wedge\ldots\wedge r'.B_u=r''.B_u\Leftrightarrow h(r'.B_1,\ldots,r'.B_u)=h(r''.B_1,\ldots,r''.B_u)\right). \qquad (138)$$

Заметим, что обратная импликация следует из инъективности хеш-функции *h*. Из (138) непосредственно вытекает следующее свойство колоночного хеш-индекса:

$$\forall r',r''\in R\left(h(r'.B_1,\ldots,r'.B_u)\neq h(r''.B_1,\ldots,r''.B_u)\Leftrightarrow r'.B_1\neq r''.B_1\vee\ldots\vee r'.B_u\neq r''.B_u\right).$$

Фрагментация колоночного хеш-индекса осуществляется на основе доменно-интервального принципа с помощью функции фрагментации $\varphi_{I_h}:I_h\to\{0,\ldots,k-1\}$, определенной следующим образом:

$$\forall x \in I_h \left(\varphi_{I_h}(x) = \varphi_{\mathbb{Z}_{\geq 0}}(x.H) \right), \tag{139}$$

где $\varphi_{\mathbb{Z}_{\geq 0}} : \mathbb{Z}_{\geq 0} \to \{0,\ldots,k-1\}$ – доменная функция фрагментации для домена $\mathfrak{D}_H = \mathbb{Z}_{\geq 0}$.

2.7. Декомпозиция реляционных операций с использованием фрагментированных колоночных хеш-индексов

2.7.1. Пересечение

В данном разделе рассматривается декомпозиция операции пересечения вида $\pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S)$ с использование распределенных колоночных хеш-индексов. При этом предполагается, что $\pi_{B_1,\ldots,B_u}(R)$ и $\pi_{B_1,\ldots,B_u}(S)$ не содержат дубликатов.

Пусть заданы два отношения $R(A,B_1,\ldots,B_u)$ и $S(A,B_1,\ldots,B_u)$, имеющие одинаковый набор атрибутов. Пусть имеются два колоночных хеш-индекса $I_{R,h}$ и $I_{S,h}$ для атрибутов $B_1,\ldots,B_u$ отношений *R* и *S*, построенные с помощью одной и той же инъективной хеш-функции $h : \mathfrak{D}_{B_1} \times \ldots \times \mathfrak{D}_{B_u} \to \mathbb{Z}_{\geq 0}$. Пусть для этих индексов задана доменно-интервальная фрагментация степени k:

$$I_{R,h} = \bigcup_{i=0}^{k-1} I_{R,h}^i ; \tag{140}$$

$$I_{S,h} = \bigcup_{i=0}^{k-1} I_{S,h}^i . \tag{141}$$

Положим

$$P^i = \pi_{I_{R,h}^i.A \to A_R,\ I_{S,h}^i.A \to A_S} \left(I_{R,h}^i \underset{(I_{R,h}^i.H = I_{S,h}^i.H)}{\bowtie} I_{S,h}^i \right) \tag{142}$$

для всех $i = 0,\ldots,k-1$. Определим

$$P = \bigcup_{i=0}^{k-1} P^i \,. \tag{143}$$

Положим

$$Q = \left\{ \&_R \left(p.A_R \right) \mid p \in P \right\}. \tag{144}$$

Теорема 10. $\pi_{B_1,\ldots,B_u}(Q) = \pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S)$.

Доказательство. Сначала докажем, что

$$\pi_{B_1,\ldots,B_u}(Q) \subset \pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S). \tag{145}$$

Пусть

$$(a, b_1, \ldots, b_u) \in Q. \tag{146}$$

Из (144) следует, что

$$(a, b_1, \ldots, b_u) = r \in R \tag{147}$$

и

$$a = r.A \in \pi_{A_R}\left(P\right). \tag{148}$$

Отсюда следует, что существует $p \in P$ такой, что $p.A_R = a \wedge p.A_S = a'$. С учетом (143) получаем, что существует i, для которого $p \in P^i$. С учетом (142) отсюда получаем, что при некотором $\chi \in \mathbb{Z}_{\geq 0}$:

$$(a, \chi) \in I^i_{R,h} \subset I_{R,h};$$
$$(a', \chi) \in I^i_{S,h} \subset I_{S,h}.$$

Поскольку хеш-функция h является инъективной, то для нее существует обратная функция. Положим $(b'_1, \ldots, b'_u) = h^{-1}(\chi)$. Тогда по определению колоночного хеш-индекса имеем

$$(a, b'_1, \ldots, b'_u) \in R;$$
$$(a', b'_1, \ldots, b'_u) \in S.$$

Поскольку A является суррогатным ключом в R, с учетом (147) получаем

$$(a, b_1, \ldots, b_u) \in R;$$
$$(a', b_1, \ldots, b_u) \in S.$$

Следовательно $(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S)$, и (145) имеет место.

Теперь докажем, что

$$\pi_{B_1,\ldots,B_u}(R) \cap \pi_{B_1,\ldots,B_u}(S) \subset \pi_{B_1,\ldots,B_u}(Q). \quad (149)$$

Пусть

$$(a,b_1,\ldots,b_u) = r \in R \quad (150)$$

и

$$(a',b_1,\ldots,b_u) = s \in S. \quad (151)$$

Положим $\chi = h(b_1,\ldots,b_u)$. Тогда по определению колоночного хеш-индекса

$$(a,\chi) \in I_{R,h};$$
$$(a',\chi) \in I_{S,h}.$$

В силу свойства (10) доменно-интервальной фрагментации существует i такой, что

$$(a,\chi) \in I^i_{R,h};$$
$$(a',\chi) \in I^i_{S,h}.$$

На основе (142) и (143) отсюда получаем $(a,a') \in P^i \subset P$. Поскольку A – суррогатный ключ в R, с учетом (144) имеем $(a,b_1,\ldots,b_u) \in Q$ и следовательно $(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(Q)$. Таким образом (149) имеет место. *Теорема доказана*.

2.7.2. Объединение

Пусть заданы два отношения $R(A,B_1,\ldots,B_u)$ и $S(A,B_1,\ldots,B_u)$. Выполним декомпозицию операции объединения вида $\pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S)$. Пусть имеются два колоночных хеш-индекса $I_{R,h}$ и $I_{S,h}$ для атрибутов $B_1,\ldots,B_u$ отношений *R* и *S*, построенные с помощью одной и той же инъективной хеш-функции $h: \mathfrak{D}_{B_1} \times \ldots \times \mathfrak{D}_{B_u} \to \mathbb{Z}_{\geq 0}$. Пусть для этих индексов задана доменно-интервальная фрагментация степени k:

$$I_{R,h} = \bigcup_{i=0}^{k-1} I_{R,h}^{i}; \tag{152}$$

$$I_{S,h} = \bigcup_{i=0}^{k-1} I_{S,h}^{i}. \tag{153}$$

Положим

$$P_i = \pi_A\left(I_{S,h}^{i}\right) \setminus \pi_{I_{S,h}^{i}.A}\left(I_{R,h}^{i} \underset{(I_{R,h}^{i}.H = I_{S,h}^{i}.H)}{\bowtie} I_{S,h}^{i}\right) \tag{154}$$

для всех $i = 0,\ldots,k-1$.

Определим

$$P = \bigcup_{i=0}^{k-1} P_i \ . \tag{155}$$

Положим

$$Q = \left\{ \&_S\left(p.A\right) \mid p \in P \right\}. \tag{156}$$

Теорема 11. $\pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(Q) = \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S)$.

Доказательство. Сначала докажем, что

$$\pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(Q) \subset \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S). \tag{157}$$

Пусть

$$(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(Q). \tag{158}$$

Тогда как минимум одно из следующих условий истинно:

$$(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(R); \tag{159}$$

$$(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(Q). \tag{160}$$

Если имеет место (159), то очевидно $(b_1,\ldots,b_u) \in \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S)$ и (157) имеет место.

Пусть истинно условие (160). Тогда из (156) следует, что существует $p \in P$ такой, что

$$\&_S\left(p.A\right) = s \in S$$

и

$$s = (a, b_1, \ldots, b_u),$$

где $a = p.A$. Отсюда получаем $(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(S)$, то есть $(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S)$ и (157) также имеет место.

Теперь докажем, что

$$\pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(Q) \supset \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S). \quad (161)$$

Пусть

$$(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(S). \quad (162)$$

Тогда как минимум одно из следующих условий истинно:

$$(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(R); \quad (163)$$

$$(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(S). \quad (164)$$

Если имеет место (163), то очевидно $(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(R) \cup \pi_{B_1,\ldots,B_u}(Q)$ и (161) имеет место.

Пусть истинно условие (164) и

$$(b_1, \ldots, b_u) \notin \pi_{B_1,\ldots,B_u}(R). \quad (165)$$

Тогда существует

$$s = (a, b_1, \ldots, b_u) \in S. \quad (166)$$

Положим

$$\chi = h(b_1, \ldots, b_u). \quad (167)$$

Тогда по определению колоночного хеш-индекса

$$(a, \chi) \in I_{S,h}.$$

В силу свойства (10) доменно-интервальной фрагментации существует $i \in \{0, \ldots, k-1\}$ такой, что

$$(a, \chi) \in I^i_{S,h}.$$

Покажем, что

$$(a) \notin \pi_{I^i_{S,h}.A}\left(I^i_{R,h} \underset{(I^i_{R,h}.H = I^i_{S,h}.H)}{\bowtie} I^i_{S,h} \right) \quad (168)$$

Предположим противное, то есть

$$(a) \in \pi_{I^i_{S,h}.A}\left(I^i_{R,h} \underset{(I^i_{R,h}.H = I^i_{S,h}.H)}{\bowtie} I^i_{S,h} \right) \tag{169}$$

Тогда существует $(a') \in \pi_{I^i_{R,h}.A}\left(I^i_{R,h}\right)$ такой, что $(a', \chi) \in I^i_{R,h}$. С учетом (167) в силу инъективности хеш-функции h получаем $(a', b_1, \ldots, b_u) \in R$, то есть $(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(R)$. Получили противоречие с (165). Значит (168) имеет место. Из (168) и (154) следует, что $(a) \in P_i$, и из (155) следует, что $(a) \in P$. Тогда из (156) следует, что существует $s' = (a, b'_1, \ldots, b'_u) \in Q \subseteq S$. Так как A – суррогатный ключ, сопоставляя это с (166), получаем $(a, b_1, \ldots, b_u) = (a, b'_1, \ldots, b'_u) \in Q$, откуда $(b_1, \ldots, b_u) \in \pi_{B_1,\ldots,B_u}(Q)$. Таким образом (161) имеет место. *Теорема доказана*.

2.7.3. Естественное соединение

Пусть заданы два отношения

$$R\left(A, B_1, \ldots, B_u, C_1, \ldots, C_v\right)$$

и

$$S\left(A, B_1, \ldots, B_u, D_1, \ldots, D_w\right).$$

Пусть имеются два колоночных хеш-индекса $I_{R,h}$ и $I_{S,h}$ для атрибутов $B_1, \ldots, B_u$ отношений *R* и *S*, построенные с помощью одной и той же инъективной хеш-функции $h : \mathfrak{D}_{B_1} \times \ldots \times \mathfrak{D}_{B_u} \to \mathbb{Z}_{\geq 0}$. Пусть для этих индексов задана доменно-интервальная фрагментация степени k:

$$I_{R,h} = \bigcup_{i=0}^{k-1} I^i_{R,h}; \tag{170}$$

$$I_{S,h} = \bigcup_{i=0}^{k-1} I^i_{S,h}. \tag{171}$$

Положим

$$P^i = \pi_{I^i_{R,h}.A \to A_R,\ I^i_{S,h}.A \to A_S}\left(I^i_{R,h} \underset{(I^i_{R,h}.H = I^i_{S,h}.H)}{\bowtie} I^i_{S,h} \right) \tag{172}$$

для всех $i = 0,\ldots,k-1$. Определим

$$P = \bigcup_{i=0}^{k-1} P^i . \tag{173}$$

Построим отношение $Q(B_1,\ldots,B_u,C_1,\ldots,C_v,D_1,\ldots,D_w)$ следующим образом:

$$\begin{aligned} Q = \{ & (\&_R(p.A_R).B_1,\ldots,\&_R(p.A_R).B_u, \\ & \&_B(p.A_B).C_1,\ldots,\&_B(p.A_B).C_v, \\ & \&_S(p.A_S).D_1,\ldots,\&_S(p.A_S).D_w) \mid p \in P \}. \end{aligned} \tag{174}$$

Теорема 12. $Q = \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$.

Доказательство. Сначала докажем, что

$$Q \subset \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S). \tag{175}$$

Пусть

$$(b_1,\ldots,b_u,c_1,\ldots,c_v,d_1,\ldots,d_w) \in Q. \tag{176}$$

Из (174) следует, что существуют кортежи *r* и *s* такие, что

$$(a,b_1,\ldots,b_u,c_1,\ldots,c_v) = r \in R, \tag{177}$$

$$(a',b_1',\ldots,b_u',d_1,\ldots,d_w) = s \in S \tag{178}$$

и

$$(r.A,s.A) = p \in P. \tag{179}$$

С учетом (173) получаем, что существует *i*, для которого $p \in P^i$. С учетом (172) отсюда получаем, что при некотором $\chi \in \mathbb{Z}_{\geq 0}$:

$$(a,\chi) \in I^i_{R,h} \subset I_{R,h};$$
$$(a',\chi) \in I^i_{S,h} \subset I_{S,h}.$$

Поскольку хеш-функция *h* является инъективной, то для нее существует обратная функция. Положим $(b_1'',\ldots,b_u'') = h^{-1}(\chi)$. Тогда по определению колоночного хеш-индекса имеем

$$(a, b_1'', \ldots, b_u'') \in \pi_{A,B_1,\ldots,B_u}(R); \quad (180)$$

$$(a', b_1'', \ldots, b_u'') \in \pi_{A,B_1,\ldots,B_u}(S). \quad (181)$$

Поскольку A является суррогатным ключом в R и S, из (177), (178), (180) и (181) следует $b_1 = b_1'' = b_1', \ldots, b_u = b_u'' = b_u'$, то есть

$$(a, b_1, \ldots, b_u, c_1, \ldots, c_v) \in R,$$
$$(a', b_1, \ldots, b_u, d_1, \ldots, d_w) \in S.$$

Следовательно $(b_1, \ldots, b_u, c_1, \ldots, c_v, d_1, \ldots, d_w) \in \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S)$, и (175) имеет место.

Теперь докажем, что

$$Q \supset \pi_{*\setminus A}(R) \bowtie \pi_{*\setminus A}(S). \quad (182)$$

Пусть

$$(a, b_1, \ldots, b_u, c_1, \ldots, c_v) \in R \quad (183)$$

и

$$(a', b_1, \ldots, b_u, d_1, \ldots, d_w) \in S. \quad (184)$$

Положим $\chi = h(b_1, \ldots, b_u)$. Тогда по определению колоночного хеш-индекса

$$(a, \chi) \in I_{R,h};$$
$$(a', \chi) \in I_{S,h}.$$

В силу свойства (10) доменно-интервальной фрагментации существует $i \in \{0, \ldots, k-1\}$ такой, что

$$(a, \chi) \in I_{R,h}^i;$$
$$(a', \chi) \in I_{S,h}^i.$$

На основе (172) и (173) отсюда получаем $(a, a') \in P^i \subset P$. Поскольку A – суррогатный ключ в R и S, с учетом (174), (183) и (184) имеем $(b_1, \ldots, b_u, c_1, \ldots, c_v, d_1, \ldots, d_w) \in Q$. Таким образом (182) имеет место. *Теорема доказана*.

2.8. Выводы по главе 2

Предложенные в главе 2 доменно-колоночная модель и распределенные колоночные индексы позволяют выполнить декомпозицию всех основных реляционных операций на подоперации, выполнение которых не требует обменов данными. Формальное описание доменно-колоночной модели дает возможность провести математические доказательства корректности предложенных методов декомпозиции. Результаты, описанные в этой главе, опубликованы в работах [2, 3, 6, 7, 9, 10, 76].

ГЛАВА 3. КОЛОНОЧНЫЙ СОПРОЦЕССОР КСОП

На базе описанной в главе 2 доменно-колоночной модели представления данных и методов декомпозиции реляционных операций в рамках диссертационного исследования разработана программная система «Колоночный СОПроцессор (КСОП)» (Columnar COProcessor CCOP) для кластерных вычислительных систем. В данной главе описывается архитектура, процесс проектирования и реализации колоночного сопроцессора КСОП.

3.1. Системная архитектура

Колоночный сопроцессор КСОП – это программная система, предназначенная для управления распределенными колоночными индексами, размещенными в оперативной памяти кластерной вычислительной системы. Назначение КСОП – вычислять таблицы предварительных вычислений для ресурсоемких реляционных операций по запросу СУБД. Общая схема взаимодействия СУБД и КСОП изображена на рис. 7.

КСОП включает в себя программу «*Координатор*», запускаемую на узле вычислительного кластера с номером 0, и программу «*Исполнитель*», запускаемую на всех остальных узлах, выделенных для работы КСОП. На SQL-сервере устанавливается специальная программа «Драйвер КСОП», обеспечивающая взаимодействие с координатором КСОП по протоколу TCP/IP.

КСОП работает только с данными целых типов 32 или 64 байта. При создании колоночных индексов для атрибутов других типов, их значение кодируется в виде целого числа, или вектора целых чисел. В последнем случае длина вектора является фиксированной и называется *размерностью колоночного индекса*.

КСОП поддерживает следующие основные операции, доступные СУБД через интерфейс драйвера КСОП.

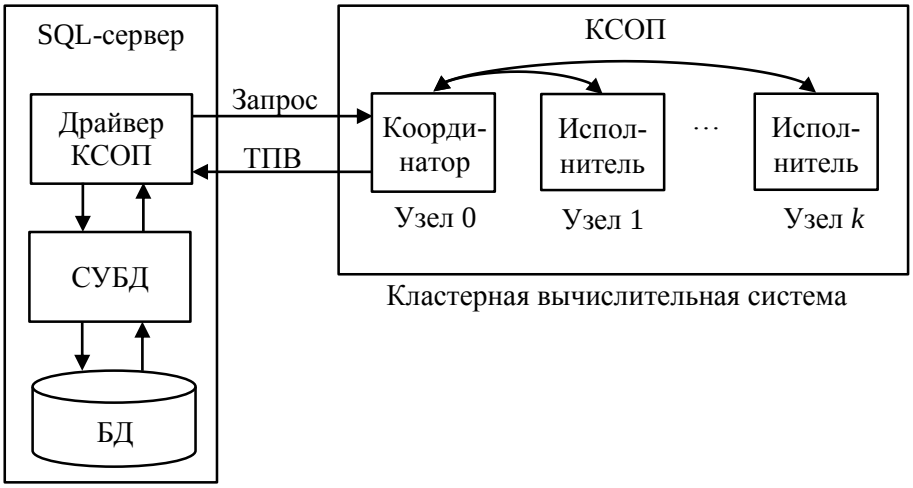


Рис. 7. Взаимодействие SQL-сервера с КСОП.

- CreateColumnIndex(TableID, ColumnID, SurrogateID, Width, Bottom, Top, Dimension) – создание распределенного колоночного индекса для атрибута ColumnID отношения TableID с параметрами: SurrogateID – идентификатор суррогатного ключа, Width – разрядность (32 или 64 бита); Bottom, Top – нижняя и верхняя границы доменного интервала; Dimension – размерность колоночного индекса. Возвращаемое значение: CIndexID – идентификатор созданного колоночного индекса.
- Insert(CIndexID, SurrogateKey, Value[*]) – добавление в колоночный индекс CIndexID нового кортежа (SurrogateKey, Value[*]).
- TransitiveInsert(CIndexID, SurrogateKey, Value[*], TValue[*]) – добавление в колоночный индекс CIndexID нового кортежа (SurrogateKey, Value[*]) с фрагментацией и сегментацией, определяемыми значением TValue[*].
- Delete(CIndexID, SurrogateKey, Value[*]) – удаление из колоночного индекса кортежа (SurrogateKey, Value[*]).
- TransitiveDelete(TCIndexID, SurrogateKey, TransitiveValue[*]) – удаление из колоночного индекса кортежа (SurrogateKey, Value[*]) с фрагментацией и сегментацией, определяемыми значением TValue[*].

- Execute(Query) – выполнение запроса на вычисление ТПВ с параметрами: Query – символьная строка, содержащая запрос в формате JSON. Примеры запросов приведены в разделе 3.2. Возвращаемое значение: PCTID – идентификатор Таблицы предварительных вычислений.

Взаимодействие между драйвером и КСОП осуществляется путем обмена сообщениями в формате JSON. Структура основных сообщений описана в разделе 3.2.

Каждый колоночный индекс делится на фрагменты, которые в свою очередь делятся на сегменты. Все сегменты одного фрагмента располагаются в сжатом виде в оперативной памяти одного процессорного узла. Более детально управление данными в КСОП рассматривается в разделе 3.3.

Общая логика работы КСОП поясняется на простом примере в разделе 3.4.

Следует отметить, что предметом данной диссертации являлась разработка колоночного сопроцессора КСОП. Вопросы разработки драйвера КСОП выходят за рамки настоящего диссертационного исследования.

3.2. Язык CCOPQL

Для организации взаимодействия между драйвером и КСОП в ходе выполнения диссертационного исследования был разработан язык CCOPQL (CCOP Query Language), базирующийся на формате данных JSON. В данном разделе приводятся синтаксические спецификации языка CCOPQL и примеры его использования. Описание синтаксиса операторов CCOPQL выполнено с помощью языка JSON Schema [120]. Это позволяет использовать готовые библиотеки для валидации операторов языка CCOPQL.

С каждым оператором языка CCOPQL связывается уникальный целочисленный код (см. табл. 1), который указывается в свойстве "opcode" при описании схемы оператора.

Табл. 1. Коды операторов CCOPQL.

Код оператора (opcode)	Функция драйвера КСОП, порождающая оператор	Семантика оператора
1	CreateColumnIndex	Создание распределенного колоночного индекса
2	CreateTransitiveIndex	Создание транзитивно колоночного индекса
3	Execute	Выполнение запроса на вычисление ТПВ
4	Insert	Добавление кортежа в колоночный индекс
5	InsertBlock	Добавление блока кортежей в колоночный индекс
6	Update	Обновление значений кортежа в колоночном индексе
7	Delete	Удаление кортежа из колоночного индекса

3.2.1. Создание распределенного колоночного индекса

Оператор создания распределенного колоночного индекса описывается с помощью следующей схемы.

```
{   "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "CreateColumnIndex",
    "description": "Создание распределенного колоночного индекса",
    "type": "object",
    "properties": {
        "opcode": {
            "description": "Код оператора",
            "enum": [1]
        }
        "params": {
            "type": "object",
            "properties": {
                "CIndexID": {
                    "description": "Идентификатор индекса",
                    "type": "integer"
                }
                "Width": {
                    "description": "Разрядность",
                    "enum": [32, 64]
                }
                "Bottom": {
                    "description": Нижняя граница доменного интервала",
                    "type": "integer"
                }
                "Top": {
                    "description": Верхняя граница доменного интервала",
                    "type": "integer"
                }
                "Dimension": {
                    "description": "Размерность колоночного индекса",
```

```
                "type": "integer"
            }
        }
    }
}
```

Приведем пример использования данного оператора для создания распределенного колоночного индекса $I_{ORDERS.ID\_CUSTOMER}(A, ID\_CUSTOMER)$ для тестовой базы данных, описанной в разделе 4.1, при масштабном коэффициенте базы данных SF = 1.

```
{
    "opcode": 1,
    "params": {
        "CIndexID": 1,
        "Width": 32,
        "Bottom": 1,
        "Top": 630000,
        "Dimension": 1
        }
}
```

В качестве параметра CIndexID указывается целое число без знака, представляющее собой уникальный идентификатор колоночного индекса, генерируемый драйвером КСОП при его создании. Этот индекс сохраняется в словаре драйвера КСОП и в локальных словарях координатора и исполнителей, и служит для идентификации индекса при дальнейшем его использовании.

3.2.2. Создание транзитивного колоночного индекса

Оператор создания транзитивно распределенного колоночного индекса описывается с помощью следующей схемы.

```
{   "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "CreateTransitiveIndex",
    "description": "Создание транзитивно распределенного колоночного индекса",
    "type": "object",
    "properties": {
        "opcode": {
            "description": "Код операции",
            "enum": [2]
        }
        "params": {
```

```
            "type": "object",
            "properties": {
                "CIndexID": {
                    "description": "Идентификатор индекса",
                    "type": "integer"
                }
                "BaseCIndexID": {
                    "description": "Базовый колоночный индекс",
                    "type": "integer"
                }
                "Width": {
                    "description": "Разрядность",
                    "enum": [32, 64]
                }
                "Dimension": {
                    "description": "Размерность колоночного индекса",
                    "type": "integer"
                }
            }
        }
    }
}
```

Приведем пример использования данного оператора для создания транзитивного распределенного колоночного индекса $I_{ORDER.TOTALPRICE}(A, TOTALPRICE)$ для тестовой базы данных, описанной в разделе 4.1, при масштабном коэффициенте базы данных SF = 1.

```
{
    "opcode": 2,
    "params": {
        "CIndexID": 2,
        "BaseCIndexID": 1,
        "Width": 32,
        "Bottom": 1,
        "Top": 100000,
        "Dimension": 1
    }
}
```

3.2.3. Выполнение запроса на вычисление ТПВ

Оператор выполнения запроса на вычисление ТПВ описывается с помощью следующей схемы:

```
{   "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Execute",
    "description": "Выполнение запроса на вычисление ТПВ",
    "type": "object",
    "properties": {
        "opcode": {
            "description": "Код операции",
            "enum": [3]
        }
        "queryPlan": {
            "description": "План выполнения запроса",
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "nodeID": {
                        "description": "Идентификатор узла в дереве плана",
                        "type": "integer"
                    },
                    "nodeType": {
                        "description": "Тип узла в дереве плана",
                        "enum": ["leaf", "inner", "root"]
                    }
                    "indexID": {
                        "description": "Идентификатор индекса (указывается для узлов
                        типа leaf)",
                        "type": "integer"
                    }
                    "leftSon": {
                        "description": "Номер левого сына (указывается для узлов типа inner)",
                        "type": "integer"
                    }
                    "rightSon": {
                        "description": " Номер правого сына" (при наличии указывается
                        для узлов типа inner)",
                        "type": "integer"
                    }
                    "relOpCode": {
                        "description": "Код реляционной операции",
                        "enum": ["projection", "selection", "equijoin", "union",
                        "intersection", "grouping", "duplicate_elimination"]
                    },
                    "parameters": {
                        "description": "Параметры реляционной операции",
                        "type": "string"
                    }
                }
            }
        }
    }
}
```

Эта схема включает в себя свойство queryPlan, описывающее план выполнения запроса. В данном описании различаются узлы трех типов: листья, внутренние узлы и корень дерева. Для листа необходимо указать идентификатор входного отношения (колоночного индекса). Для корня и внутреннего узла указываются порядковые номера левого и правого (при наличии) сыновей в дереве плана. Корень отличается тем, что у него нет предшественников. Приведем пример использования оператора Execute для вычисления таблицы предварительных вычислений *P*(*A_ORDERS*, *A_CUSTOMER*), задаваемой выражением реляционной алгебры, приведенном на стр. 106. Соответствующий план выполнения запроса изображен на рис. 8. Этот план строится драйвером КСОП по следующим правилам. Атрибуты исходных и промежуточных отношений нумеруются натуральными числами слева-направо.

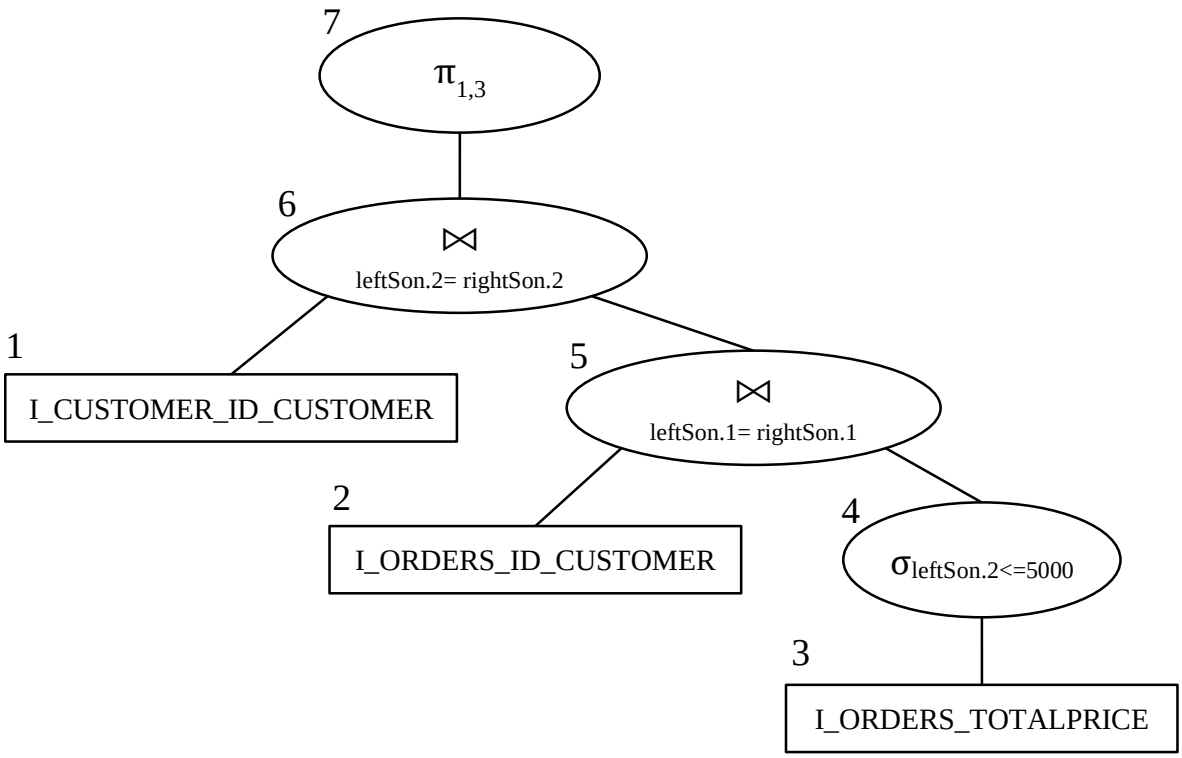


Рис. 8. Дерево плана запроса.

Для унарных операций непустым всегда является ссылка на левого сына в дереве плана. Обозначение leftSon.1 именует значение первого слева атрибута отношения, вычисляемого левым сыном. Аналогично, rightSon.2 именует значение второго слева атрибута отношения, вычисляемого правым сыном. Все узлы дерева плана нумеруются в порядке обратного обхода дерева (левый сын, правый сын, корень). В соответствие с этим драйвер сгенерирует следующее описание оператора Execute (для случая *Sel* = 0.05):


```
{   "opcode": 3,
    "queryPlan": [
        {
            "nodeID": 1,
            "nodeType": "leaf",
            "indexID": I_CUSTOMER_ID_CUSTOMER
        },
        {
            "nodeID": 2,
            "nodeType": "leaf",
            "indexID": I_ORDERS_ID_CUSTOMER
        },
        {
            "nodeID": 3,
            "nodeType": "leaf",
            "indexID": I_ORDERS_TOTALPRICE
        },
        {
            "nodeID": 4,
            "nodeType": "inner",
            "leftSon": 3,
            "relOpCode": "selection",
            "parameters": "leftSon.2<=5000"
        },
        {
            "nodeID": 5,
            "nodeType": "inner",
            "leftSon": 2,
            "rightSon": 4,
            "relOpCode": "equijoin",
            "parameters": "leftSon.1= rightSon.1"
        },
        {
            "nodeID": 6,
            "nodeType": "inner",
            "leftSon": 1,
            "rightSon": 5,
            "relOpCode": "equijoin",
            "parameters": "leftSon.2=rightSon.2"
        },
```

```
        {   "nodeID": 7,
            "nodeType": "root",
            "leftSon": 6,
            "relOpCode": "projection",
            "parameters": "1, 3"
        }
    ]
}
```

После окончания обработки запроса КСОП передает драйверу таблицу предварительных вычислений в сжатом виде. Драйвер обеспечивает доступ к ТПВ для SQL-сервера. Вопросы организации такого доступа выходят за рамки настоящего диссертационного исследования.

3.2.4. Добавление кортежа в колоночный индекс

Оператор добавления кортежа в колоночный индекс описывается с помощью следующей схемы:

```
{   "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Insert",
    "description": "Добавление в колоночный индекс нового кортежа",
    "type": "object",
    "properties": {
        "opcode": {
            "description": "Код операции",
            "enum": [4]
        }
        "params": {
            "type": "object",
            "properties": {
                "CIndexID": {
                    "description": "Идентификатор колоночного индекса",
                    "type": "integer"
                }
                "SurrogateKey": {
                    "description": "Суррогатный ключ нового кортежа",
                    "type": "integer"
                }
                "Value": {
                    "description": "Значение нового кортежа",
                    "type": "array",
                    "items": { "type": "integer" }
                }
            }
        }
    }
}
```

Приведем пример использования данного оператора для добавления кортежа в колоночный индекс $I_{ORDERS.ID\_CUSTOMER}(A, ID\_CUSTOMER)$ для тестовой базы данных, описанной в разделе 4.1.

```
{
   "opcode": 4,
   "params": {
      "CIndexID": 1,
      "SurrogateKey": 0,
      "Value": [38335];
   }
}
```

3.2.5. Добавление блока кортежей в колоночный индекс

Оператор добавления блока кортежей в колоночный индекс описывается с помощью следующей схемы:

```
{   "$schema": "http://json-schema.org/draft-04/schema#",
   "title": "InsertBlock",
   "description": "Добавление блока кортежей в колоночный индекс",
   "type": "object",
   "properties": {
      "opcode": {
         "description": "Код операции",
         "enum": [5]
      }
      "params": {
         "type": "object",
         "properties": {
            "TupleBlock": {
               "description": "Блок кортежей",
               "type": "array",
               "items": {
                  "type": "object",
                  "properties": {
                        "SurrogateKey": {
                                 "description": "Суррогатный ключ нового кортежа",
                                 "type": "integer"
                        },
                        "Value": {
                                 "description": "Значение нового кортежа",
                                 "type": "array",
                                 "items": { "type": "integer" }
                        }
```

```
                }
            }
            "BlockSize": {
                "description": "Количество кортежей в блоке",
                "type": "integer"
            }
        }
    }
}
```

Приведем пример использования данного оператора для добавления блока кортежей в колоночный индекс $I_{ORDERS.ID\_CUSTOMER}(A, ID\_CUSTOMER)$ для тестовой базы данных, описанной в разделе 4.1.

```
{
    "opcode": 5,
    "params": {
        "CIndexID": 1,
        "TupleBlock": [
            {"SurrogateKey": 1, "Value": [1220]},
            {"SurrogateKey": 2, "Value": [580]},
            {"SurrogateKey": 3, "Value": [621056]}
        ]
        "BlockSize": 3;
    }
}
```

3.2.6. Обновление значений кортежа в колоночном индексе

Оператор обновления значений кортежа в колоночном индексе описывается с помощью следующей схемы:

```
{   "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Update",
    "description": "Обновление значений кортежа в колоночном индексе",
    "type": "object",
    "properties": {
        "opcode": {
            "description": "Код операции",
            "enum": [6]
        }
        "params": {
            "type": "object",
```

```
        "properties": {
            "CIndexID": {
                "description": "Идентификатор колоночного индекса",
                "type": "integer"
            }
            "SurrogateKey": {
                "description": "Суррогатный ключ обновляемого кортежа",
                "type": "integer"
            }
            "NewValue": {
                "description": "Новое значение кортежа",
                "type": "array",
                    "items": {
                        "type": "integer"
                    }
                }
            }
        }
    }
}
```

Приведем пример использования данного оператора для обновления кортежа в колоночном индексе $I_{ORDERS.ID\_CUSTOMER}(A, ID\_CUSTOMER)$ для тестовой базы данных, описанной в разделе 4.1.

```
{
    "opcode": 6,
    "params": {
        "CIndexID": 1,
        "SurrogateKey": 1,
        "NewValue": [7]
    }
}
```

3.2.7. Удаление кортежа из колоночного индекса

Оператор удаления кортежа из колоночного индекса описывается с помощью следующей схемы:

```
{   "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Delete",
    "description": "Удаление кортежа в колоночном индексе",
    "type": "object",
```

```
    "properties": {
        "opcode": {
            "description": "Код операции",
            "enum": [7]
        }
        "params": {
            "type": "object",
            "properties": {
                "CIndexID": {
                    "description": "Идентификатор колоночного индекса",
                    "type": "integer"
                }
                "SurrogateKey": {
                    "description": "Суррогатный ключ удаляемого кортежа",
                    "type": "integer"
                }
                "Value": {
                    "description": "Значение удаляемого кортежа",
                    "type": "array",
                        "items": {
                            "type": "integer"
                        }
                    }
                }
            }
        }
    }
}
```


Приведем пример использования данного оператора для удаления кортежа в колоночном индексе $I_{ORDERS.ID\_CUSTOMER}$(*A, ID_CUSTOMER*) для тестовой базы данных, описанной в разделе 4.1.


```
{
    "opcode": 7,
    "params": {
        "CIndexID": 1,
        "SurrogateKey": 2,
        "Value": [580]
    }
}
```

3.3. Управление данными

Все данные (колоночные индексы и метаданные), с которыми работает КСОП, хранятся в распределенной памяти кластерной вычислительной системы. Для колоночных индексов поддерживается двухуровневая система их разбиения на непересекающиеся части. Поясним ее работу на примере, изображенном на рис. 9, где показано разбиение колоночного индекса, построенного для атрибута *B*. В основе разбиения лежит домен $\mathfrak{D}_B = [0,99]$, на котором определен атрибут *B*. Домен разбивается на *сегментные интервалы*

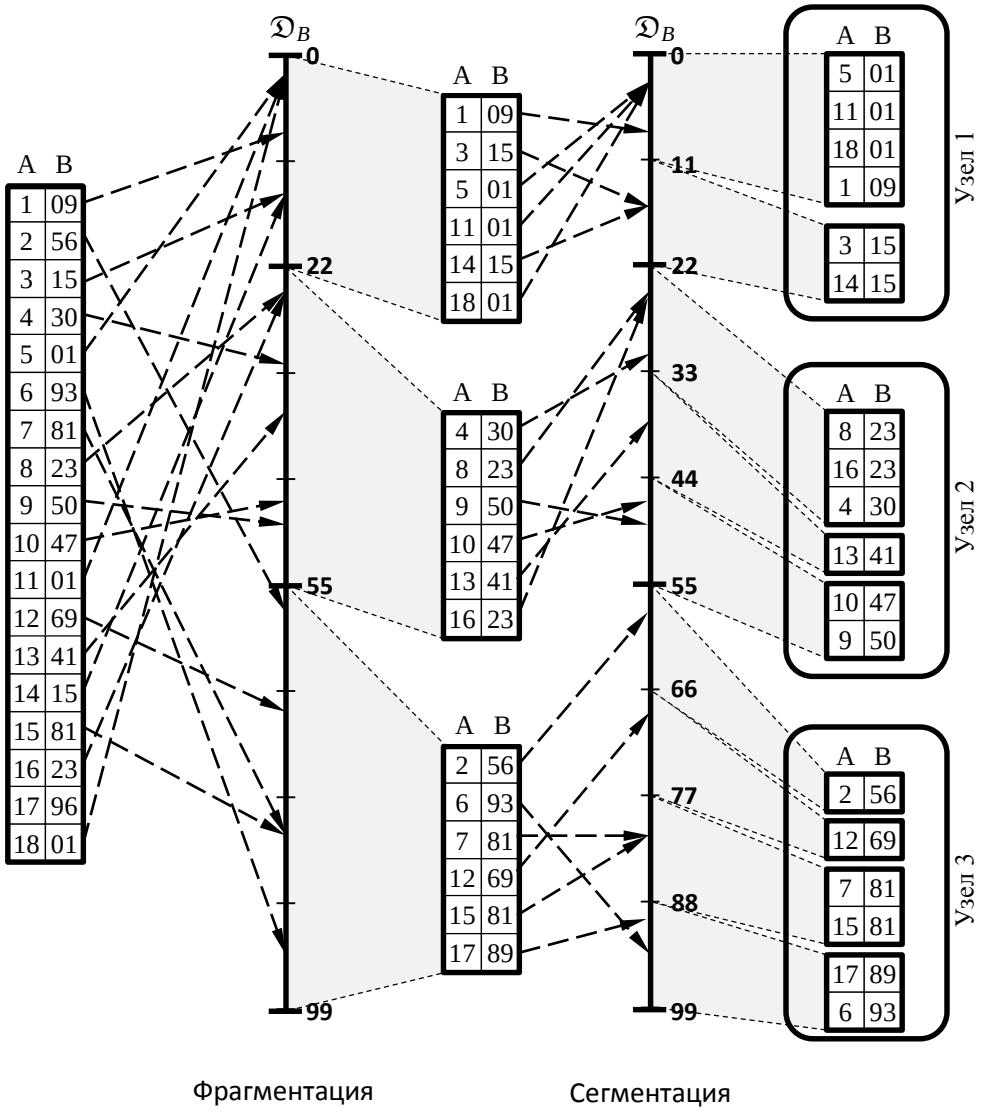


Рис. 9. Пример двухуровневого разбиения колоночного индекса на фрагменты и сегменты.

равной длины: [0;11), [11;22), …, [77;88), [88;99]. Количество сегментных интервалов должно превышать суммарное количество процессорных ядер на узлах-исполнителях (см. раздел 4.2). Сегментные интервалы разбиваются на последовательные группы, которые называются *фрагментными интервалами*. В примере на рис. 9 это следующие интервалы: [0;22), [22;55), [55;99]. Количество фрагментных интервалов должно совпадать с количеством узлов-исполнителей. Длины фрагментных интервалов могут не совпадать. Это необходимо для балансировки загрузки процессорных узлов в условиях перекоса данных.

На первом этапе распределения данных исходный колоночный индекс разбивается на фрагменты. Каждому фрагменту соответствует определенный фрагментный интервал. Кортеж (a, b) попадает в данный фрагмент тогда, и только тогда, когда значение b принадлежит соответствующему фрагментному интервалу. Все кортежи, принадлежащие одному фрагменту, хранятся на одном и том же процессорном узле. На втором этапе каждый фрагмент разбивается на сегменты. Каждому сегменту соответствует определенный сегментный интервал. Кортеж (a, b) попадает в данный сегмент тогда, и только тогда, когда значение b принадлежит соответствующему сегментному интервалу. Внутри каждого сегментного интервала записи сортируются в порядке возрастания значения атрибута.

В случае неравномерного распределения значений атрибута, по которому создан колоночный индекс, можно добиться примерно одинакового размера фрагментов, сдвигая границы фрагментных интервалов, как это сделано в примере на рис. 9. При этом могут получиться сегменты разной длины. В КСОП сегмент является наименьшей единицей распределения работ между процессорными ядрами. Если количество сегментов не велико по сравнению с количеством процессорных ядер на одном узле, то при выполнении запроса мы получим дисбаланс в загрузке процессорных ядер. Проблему балансировки загрузки можно эффективно решить, уменьшив длину

сегментных интервалов, и тем самым увеличив количество сегментов. Детально этот вопрос исследуется в разделе 4.2.

КСОП работает только с данными целых типов (32 или 64 бита). Данные других типов должны быть предварительно закодированы драйвером КСОП в виде последовательности целых чисел. При этом могут использоваться методы, описанные в работе [84]. В случае, когда значение атрибута кодируется в виде целочисленного вектора размерности n (например, это может быть применено для длинных символьных строк), домен превращается в n-мерный куб. В этой ситуации n-мерный куб разбивается на n-мерные параллелепипеды по числу процессорных узлов (аналог фрагментного интервала), каждый из которых разбивается на еще меньшие n-мерные параллелепипеды по числу процессорных ядер в одном узле (аналог сегментного интервала). Все построения, выполненные в главе 2, можно обобщить на n-мерные фрагментные интервалы, однако это выходит за рамки настоящего диссертационного исследования.

Для сжатия закодированных сегментов могут использоваться как «тяжеловесные» методы (например, Хаффмана [72] или Лемпеля-Зива [141]), так и «легковесные» (например, Run-Length Encoding [25, 32] или Null Suppression [113]), либо их комбинация [112]. В версии КСОП, реализованной в ходе выполнения диссертационного исследования, для сжатия сегментов использовалась библиотека Zlib [51, 111], реализующая метод сжатия DEFLATE [50], являющийся комбинацией методов Хаффмана и Лемпеля-Зива. В работе [25] было показано, что легковесные методы типа Run-Length Encoding в случае колоночного представления информации могут оказаться более эффективными, чем тяжеловесные, поскольку допускают выполнение операций над данными без их распаковки. Однако этот вопрос не вошел в план исследований по данной диссертации.

3.4. Пример выполнения запроса

Поясним общую логику работы КСОП на простом примере. Пусть имеется база данных из двух отношений *R*(*A*,*B*,*D*) и *S*(*A*,*B*,*C*), хранящихся на SQL-сервере (см. рис. 10). Пусть нам необходимо выполнить запрос:

SELECT D, C
FROM R, S
WHERE R.B = S.B **AND** C<13.

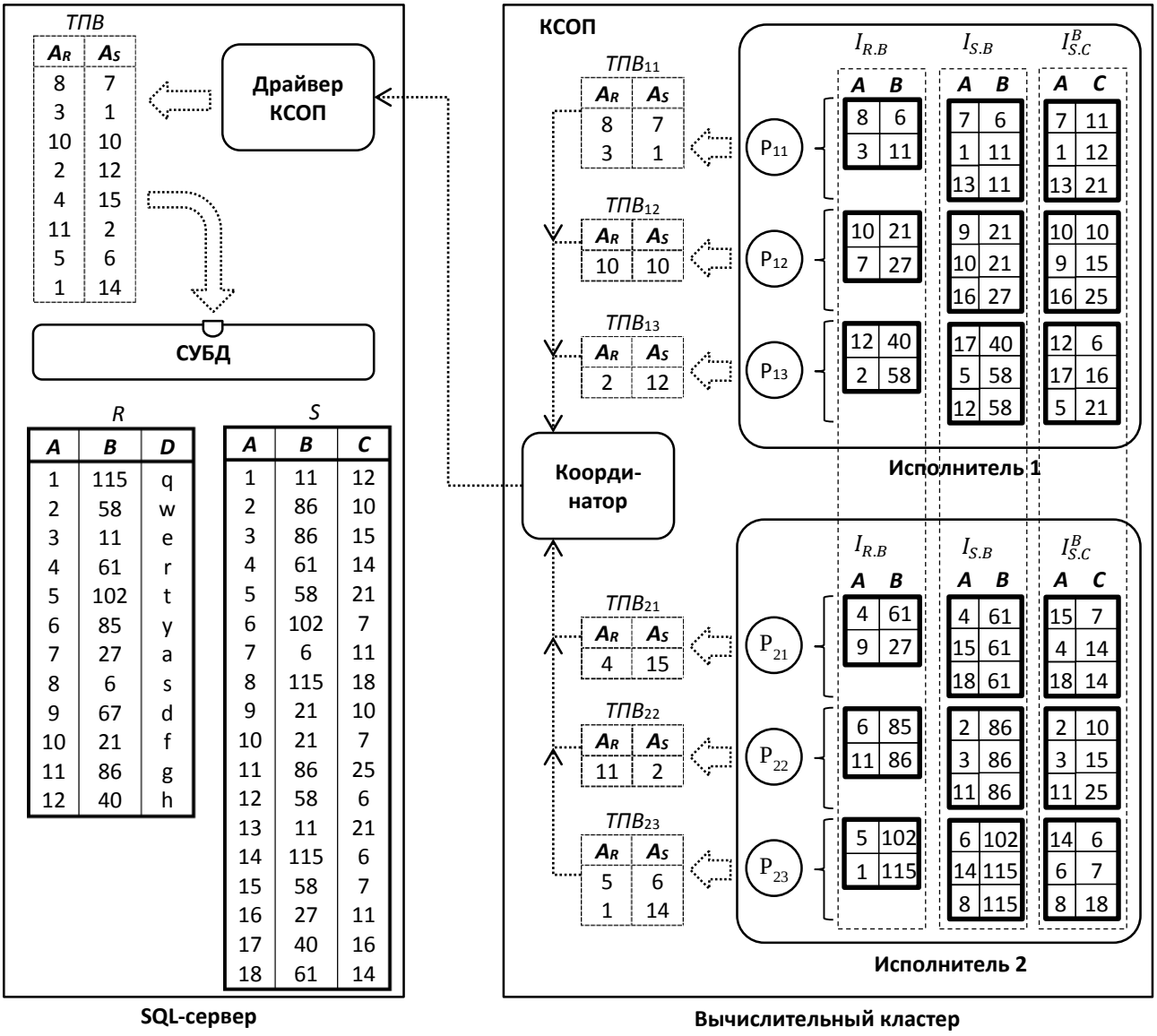


Рис. 10. Вычисление ТПВ с использованием КСОП.

Предположим, что КСОП имеет только два узла-исполнителя и на каждом узле имеется три процессорных ядра (процессорные ядра на рис. 10 промаркированы обозначениями P_{11},…, P_{23}). Положим, что атрибуты *R.B* и *S.B* определены на домене целых чисел из интервала [0; 120). Сегментные интервалы для *R.B* и *S.B* определим следующим образом: [0; 20), [20; 40), [40; 60), [60; 80), [80; 100), [100; 120). В качестве фрагментных интервалов для *R.B* и *S.B* зафиксируем: [0; 59] и [60,119]. Пусть атрибут *S.C* определен на домене целых чисел из интервала [0; 25]. Изначально администратор базы данных с помощью драйвера КСОП создает для атрибутов *R.B* и *S.B* распределенные колоночные индексы $I_{R.B}$ и $I_{S.B}$. Затем для атрибута *S.C* создается распределенный колоночный индекс $I^{B}_{S.C}$, который фрагментируется и сегментируется транзитивно относительно индекса $I_{S.B}$. Распределенные колоночные индексы $I_{R.B}$, $I_{S.B}$ и $I^{B}_{S.C}$ сохраняются в оперативной памяти узлов-исполнителей. Таким образом мы получаем распределение данных внутри КСОП, приведенное на рис. 10. При поступлении SQL-запроса, приведенного на стр. 93, он преобразуется драйвером КСОП в план, определяемый следующим выражением реляционной алгебры:

$$\pi_{I_{R.B}.A \to A_R, I_{S.B}.A \to A_S} \left(I_{R.B} \underset{\boxed{\substack{I_{R.B}.B= \\ I_{S.B}.B}}}{\bowtie} \left(I_{S.B} \bowtie \sigma_{C<13} \left(I^{B}_{S.C} \right) \right) \right).$$

При выполнении драйвером операции Execute указанный запрос передается координатору КСОП в виде оператора CCOPQL в формате JSON (см. раздел 3.2.3). Запрос выполняется независимо процессорными ядрами узлов-исполнителей над соответствующими группами сегментов. При этом за счет доменной фрагментации и сегментации не требуются обмены данными как между узлами-исполнителями, так и между процессорными ядрами одного узла. Каждое процессорное ядро вычисляет свою часть ТПВ, которая пересылается на узел-координатор. Координатор объединяет фрагменты ТПВ в

единую таблицу и пересылает ее драйверу, который выполняет материализацию этой таблицы в виде отношения в базе данных, хранящейся на SQL-сервере. После этого SQL-сервер вместо исходного SQL-оператора, приведенного на стр. 93, выполняет следующий оператор:

```
SELECT D, C
FROM
   R INNER JOIN (
      ТПВ INNER JOIN S ON (S.A = ТПВ.AS)
   ) ON (R.A = ТПВ.AR).
```

При этом используются обычные кластеризованные индексы в виде B-деревьев, заранее построенные для атрибутов *R.A* и *S.A*. После выполнения запроса ТПВ удаляется, а распределенные колоночные индексы $I_{R.B}$, $I_{S.B}$ и $I^{B}_{S.C}$ сохраняются в оперативной памяти узлов-исполнителей для последующего использования.

3.5. Проектирование и реализация

На рис. 11 представлена диаграмма развертывания КСОП, основанная на нотации стандарта UML 2.0. КСОП является распределенной системой и включает в себя два типа узлов: *координаторы* и *исполнители*. В простейшем случае в системе имеется один узел-координатор и несколько узлов-исполнителей, как это показано на рис. 11. Однако, если при большом количестве исполнителей координатор становится узким местом, в системе может быть несколько координаторов. В этом случае они образуют иерархию в виде сбалансированного дерева, листьями которого являются узлы-исполнители.

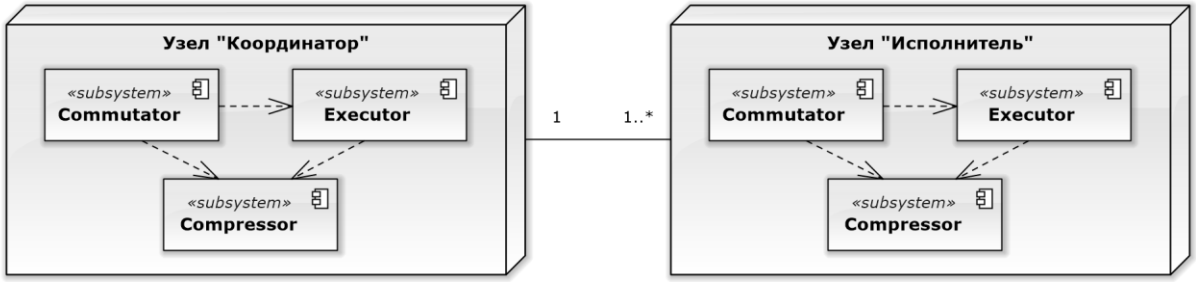


Рис. 11. Структура КСОП.

И координатор, и исполнитель имеют унифицированную структуру, состоящую из трех компонент: *Commutator* (*Коммутатор*), *Executor* (*Исполнитель*), *Compressor* (*Модуль сжатия*). Однако их реализации в координаторе и исполнителе различается. Компонент Commutator в координаторе выполняет две функции: 1) обмен сообщениями в формате JSON с драйвером КСОП по протоколу TCP/IP (см. рис. 7); 2) обмен сообщениями с исполнителями по технологии MPI. Компонент Commutator в исполнителе осуществляет обмен сообщениями с координатором по технологии MPI. Компонент Executor на исполнителе организует параллельную обработку сегментов колоночных индексов, используя технологию OpenMP, и формирует фрагмент ТПВ. Компонент Executor на координаторе объединяет фрагменты ТПВ, вычисленные исполнителями в единую таблицу. Компоненты Commutator и Executor используют компонент Compressor для сжатия/распаковки данных. Рассмотрим, как КСОП выполняет основные операции над распределенными колоночными индексами, перечисленные в разделе 3.1.

При выполнении оператора *CreateColumnIndex* координатор добавляет информацию о структуре колоночного индекса в свой локальный словарь и отправляет исполнителям указание создать в своих локальных словарях дескриптор с информацией о новом колоночном индексе. При этом координатор определяет длину сегментного интервала и границы фрагментных интервалов. Эта информация сохраняется в дескрипторе колоночного индекса. Кроме этого, дескриптор включает в себя битовую шкалу сегментов, в которой значение 1 соответствует непустым сегментам, значение 0 – пустым. При начальном создании колоночного индекса все сегменты на всех узлах-исполнителях являются пустыми.

При выполнении оператора *Insert* координатору передается кортеж (SurrogateKey, Value) для вставки в указанный колоночный индекс. Координатор определяет, в границы какого фрагментного интервала попадает зна-

чение Value, и пересылает этот кортеж на соответствующий узел-исполнитель. Исполнитель определяет номер сегментного интервала, к которому принадлежит значение Value. Если соответствующий сегмент не пуст, то кортеж вставляется в сегмент с сохранением упорядочения по полю Value. Если соответствующий сегмент пуст, то создается новый сегмент из одного кортежа.

При выполнении оператора *TransitiveInsert* координатору кроме нового кортежа (SurrogateKey, Value) передается дополнительное значение TValue, которое им используется для определения номера фрагментного интервала. Кортеж (SurrogateKey, Value) вместе со значением TValue пересылается на соответствующий узел-исполнитель. Исполнитель по значение TValue определяет номер сегментного интервала и вставляет новый кортеж в соответствующий сегмент.

При выполнении оператора *Delete* координатору передается суррогатный ключ SurrogateKey и значение Value, которые надо удалить. Координатор определяет, в границы какого фрагментного интервала попадает значение Value, и пересылает этот кортеж на соответствующий узел-исполнитель. Исполнитель определяет номер сегментного интервала, к которому принадлежит значение Value, производит в соответствующем сегменте поиск кортежа с ключом SurrogateKey и выполняет его удаление.

Оператор *TransitiveDelete* выполняется аналогично оператору *Delete* с той лишь разницей, что номера фрагментного и сегментного интервалов вычисляются по транзитивному значению TValue.

Выполнение оператора *Execute* включает в себя две фазы: 1) вычисление фрагментов ТПВ на узлах-исполнителях; 2) слияние фрагментов ТПВ в единую таблицу на узле-координаторе и пересылка ее на SQL-сервер. На первой фазе процессорные ядра (легковесные процессы Open MP) выбирают необработанные сегментные интервалы и выполняют вычисление *сегмент-*

ных ТПВ для соответствующих сегментов колоночных индексов, задействованных в запросе. После того, как все сегментные интервалы обработаны, получившийся фрагмент ТПВ пересылается на узел-координатор в сжатом виде. На второй фазе координатор распаковывает все полученные сегментные ТПВ и объединяет их в единую таблицу. Для распараллеливания этого процесса используется технология OpenMP. Получившаяся ТПВ пересылается на SQL-сервер. При этом также может использоваться сжатие данных.

Колоночный сопроцессор КСОП был реализован на языке Си с использованием аппаратно независимых параллельных технологий MPI и OpenMP. Он может работать как на ЦПУ Intel Xeon X5680, так и на сопроцессоре Intel Xeon Phi без модификации кода. Объем исходного кода составил около двух с половиной тысяч строк. Исходные тексты КСОП свободно доступны в сети Интернет по адресу: https://github.com/elena-ivanova/colomnindices/.

3.6. Выводы по главе 3

В третьей главе описана общая архитектура организации системы баз данных с использованием колоночного сопроцессора КСОП. В состав системы входит SQL-сервер и вычислительный кластер. На SQL-сервере устанавливается реляционная СУБД с внедренным коннектором и драйвер КСОП. На вычислительном кластере устанавливается колоночный сопроцессор КСОП. Все колоночные индексы создаются и поддерживаются в распределенной оперативной памяти вычислительного кластера. При создании информационных систем с такой конфигурацией необходимо будет еще подключить подсистему восстановления колоночных индексов после сбоя. Для этого копии колоночных индексов и метаданных могут создаваться на твердотельных дисках, установленных на каждом вычислительном узле. Результаты, описанные в этой главе, опубликованы в работах [2, 4].

ГЛАВА 4. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

В данной главе приводятся результаты вычислительных экспериментов по исследованию эффективности разработанных в диссертации моделей, методов и алгоритмов обработки сверхбольших баз данных с использованием распределенных колоночных индексов. Тестирование проводилось на основе колоночного сопроцессора КСОП, описанного в главе 3. В качестве тестовой базы данных использовалась синтетическая база данных, разработанная на основе теста TPC-H. Тестирование осуществлялось на двух кластерных установках: «Торнадо ЮУрГУ» и «RSC PetaStream» МСЦ РАН. В качестве СУБД, взаимодействующей с КСОП, использовалась реляционная СУБД с открытыми кодами PostgreSQL.

4.1. Вычислительная среда

Эксперименты проводились на двух вычислительных комплексах с кластерной архитектурой: «Торнадо ЮУрГУ» Южно-Уральского государственного университета и «RSC PetaStream» Межведомственного суперкомпьютерного центра Российской академии наук. Основные параметры этих систем приведены в табл. 2.

Система «*Торнадо ЮУрГУ*» [18] включает в себя 384 процессорных узлов, соединенных сетями InfiniBand QDR и Gigabit Ethernet. В состав процессорного узла входит два шестиядерных ЦПУ Intel Xeon X5680, ОЗУ 24 Гб и сопроцессор Intel Xeon Phi SE10X (61 ядро по 1.1 ГГц), соединенные шиной PCI Express.

Система «*RSC PetaStream*» [15] состоит из 8 модулей, включающих в себя 8 сопроцессоров Intel Xeon Phi 7120, каждый из которых имеет 61 процессорное ядро и 16 Гб встроенной памяти GDDR5. Модули соединены между собой сетями InfiniBand FDR и Gigabit Ethernet.

Табл. 2. Параметры вычислительных комплексов.

Параметры	**Вычислительный комплекс**	
	«Торнадо ЮУрГУ»	**«RSC PetaStream»**
Количество узлов	384	64
Тип процессоров	2 × Intel Xeon X5680 (12 ядер по 3.33 ГГц; 2 потока на ядро)	
Оперативная память узла	24 Гб	
Тип сопроцессора	Intel Xeon Phi SE10X: (61 ядро по 1.1 ГГц; 4 потока на ядро)	Intel Xeon Phi 7120 (61 ядро по 1.24 ГГц)
Память сопроцессора	8 Гб	16 Гб
Тип системной сети	InfiniBand QDR (40 Гбит/с)	InfiniBand FDR (56 Гбит/с)
Тип управляющей сети	Gigabit Ethernet	Gigabit Ethernet
Операционная система	Linux CentOS 6.2	Linux CentOS 7.0

Непосредственно на каждом сопроцессоре (на одном ядре) загружается операционная система Linux CentOS 7.0.

Для тестирования колоночного сопроцессора КСОП использовалась синтетическая база данных, построенная на основе эталонного теста TPC-H [130]. Тестовая база данных состояла из двух таблиц: **ORDERS** (ЗАКАЗЫ) и **CUSTOMER** (КЛИЕНТЫ), схема которых приведена на рис. 12. Для масштабирования базы данных использовался *масштабный коэффициент SF* (*Scale Factor*), значение которого изменялось от 1 до 10. При проведении экспериментов размер отношения **ORDERS** составлял SF×63 000 000 кортежей, размер отношения **CUSTOMER** – SF×630 000 кортежей.

Для описания типов атрибутов отношений используем следующие определения:

– *идентификатор*: уникальное в рамках столбца целое число без знака;
– *целое число*: случайное целое число из диапазона от -2 147 483 646 до 2 147 483 647;

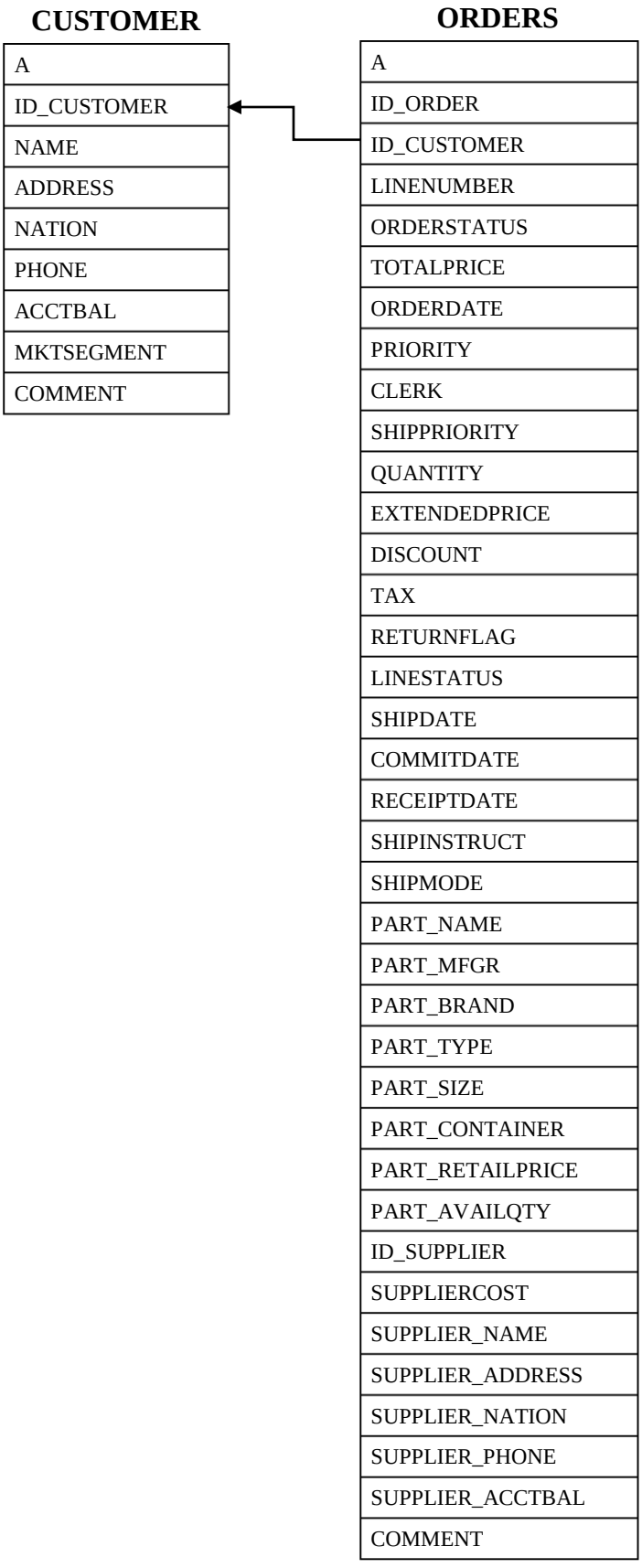


Рис. 12. Схема тестовой базы данных.

- *десятичное число*: случайное десятичное число в диапазоне от -9 999 999 999.99 до 9 999 999 999.99;
- *фиксированная символьная строка длины N*: символьная строка фиксированной длины N;
- *символьная строка длины N*: символьная строка переменной длины, максимальная длина строки равна N;
- *дата*: значение, представленное в формате YYYY-MM-DD, где YYYY: число, обозначающее год, MM: число в диапазоне от 01 до 12, обозначающее месяц, DD: число в диапазоне от 01 до 31, обозначающее день.

Отношение **CUSTOMER** состоит из следующих 9 атрибутов:

A: идентификатор, обозначающий суррогатный ключ,

ID_CUSTOMER: идентификатор, обозначающий первичный ключ,

NAME: символьная строка длины 25,

ADDRESS: символьная строка длины 40,

NATION: фиксированная символьная строка длины 25,

PHONE: фиксированная символьная строка длины 15,

ACCTBAL: десятичное число,

MKTSEGMENT: фиксированная символьная строка длины 10,

COMMENT: символьная строка длины 117.

Отношение **ORDERS** состоит из следующих 37 атрибутов:

A: идентификатор, обозначающий суррогатный ключ,

ID_ORDER: идентификатор, обозначающий первичный ключ,

ID_CUSTOMER: целое число, являющееся внешним ключом к атрибуту **CUSTOMER.ID_CUSTOMER**,

LINENUMBER: целое число,

ORDERSTATUS: фиксированная символьная строка длины 1,

TOTALPRICE: целое число в диапазоне [1;100 000],

ORDERDATE: дата,
PRIORITY: фиксированная символьная строка длины 15,
CLERK: фиксированная символьная строка длины 15,
SHIPPRIORITY: целое число,
QUANTITY: десятичное число,
EXTENDEDPRICE: десятичное число,
DISCOUNT: десятичное число,
TAX: десятичное число,
RETURNFLAG: фиксированная символьная строка длины 1,
LINESTATUS: фиксированная символьная строка длины 1,
SHIPDATE: дата,
COMMITDATE: дата,
RECEIPTDATE: дата,
SHIPINSTRUCT: фиксированная символьная строка длины 25,
SHIPMODE: фиксированная символьная строка длины 10,
PART_NAME: фиксированная символьная строка длины 55,
PART_MFGR: фиксированная символьная строка длины 25,
PART_BRAND: фиксированная символьная строка длины 10,
PART_TYPE: символьная строка длины 10,
PART_SIZE: целое число,
PART_CONTAINER: фиксированная символьная строка длины 10,
PART_RETAILPRICE: десятичное число,
PART_AVAILQTY: целое число,
ID_SUPPLIER: идентификатор,
SUPPLIERCOST: десятичное число,
SUPPLIER_NAME: фиксированная символьная строка длины 25,
SUPPLIER_ADDRESS: символьная строка длины 40,
SUPPLIER_NATION: фиксированная символьная строка длины 25,
SUPPLIER_PHONE: фиксированная символьная строка длины 15,
SUPPLIER_ACCTBAL: десятичное число,
COMMENT: фиксированная символьная строка длины 79.

Для генерации тестовой базы данных была написана программа на языке Си, использующая методику, изложенную в работе [64]. Атрибуты **CUSTOMER.A** и **ORDERS.A**, игравшие роль суррогатных ключей, заполнялись целыми числами с шагом 1, начиная со значения 0. Атрибут **CUSTOMER.ID_CUSTOMER**, являющийся первичным ключом, заполнялся целыми числами с шагом 1, начиная со значения 1. Атрибут **ORDERS.ID_ORDER**, являющийся первичным ключом, также заполнялся целыми числами с шагом 1, начиная со значения 1. Атрибут **ORDERS.ID_CUSTOMER**, являющийся внешним ключом, заполнялся значениями из интервала [1, SF*630000]. При этом, для имитирования перекоса данных использовались следующие распределения:

– равномерное (uniform);
– «45-20»;
– «65-20»;
– «80-20».

Для генерации неравномерного распределения была использована вероятностная модель. В соответствии с этой моделью коэффициент перекоса θ, $(0 \le \theta \le 1)$ задает распределение, при котором каждому целому значению из интервала [1, SF*630000] назначается некоторый весовой коэффициент $p_i (i = 1, \ldots, N)$, определяемый формулой

$$p_i = \frac{1}{i^{\theta} \cdot H_N^{(\theta)}}, \sum_{i=1}^{N} p_i = 1,$$

где N = SF*630000 – количество различных значений атрибута **ORDERS.ID_CUSTOMER** и $H_N^{(s)} = 1^{-s} + 2^{-s} + \cdots + N^{-s}$, N-е гармоническое число порядка s. В случае $\theta = 0$ распределение весовых коэффициентов соответствует равномерному распределению. При $\theta = 0.86$ распределение соответствует правилу «80-20», в соответствии с которым 20% самых популярных значений занимают 80% позиций в столбце **ID_CUSTOMER** в таблице **ORDERS**. При $\theta = 0.73$ распределение соответствует правилу «65-20» (20% самых популярных значений занимают 65% позиций в столбце

ID_CUSTOMER в таблице **ORDERS**). При $\theta = 0.5$ распределение соответствует правилу «45-20» (20% самых популярных значений занимают 45% позиций в столбце **ID_CUSTOMER** в таблице **ORDERS**).

Все остальные атрибуты в отношениях **CUSTOMER** и **ORDERS**, заполнялись случайными значениями соответствующих типов с равномерным распределением.

Тестовая база данных была развернута в СУБД PostgreSQL 9.4.0 на выделенном узле вычислительного кластера «Торнадо ЮУрГУ». В качестве тестового запроса фигурировал следующий SQL-запрос Q1:

```
# Запрос Q1
SELECT * FROM CUSTOMER, ORDERS
WHERE (CUSTOMER.ID_CUSTOMER=ORDERS.ID_CUSTOMER)
AND (ORDERS.TOTALPRICE <= Sel*100 000).
```

Для варьирования размера результирующего отношения использовался *коэффициент селективности Sel*, принимающий значение из интервала]0;1]. Коэффициент Sel определяет размер (в кортежах) результирующего отношения относительно размера отношения **ORDERS**. Например, при Sel = 0.5 размер результирующего отношения составляет 50% от размера отношения **ORDERS**, при Sel = 0.05 – 5%, а при Sel = 1 – 100%. Таким образом, большая селективность запроса соответствует меньшему значению коэффициента селективности.

С помощью колоночного сопроцессора КСОП в оперативной памяти кластерной вычислительной системы были созданы следующие распределенные колоночные индексы:

$I_{CUSTOMER.ID\_CUSTOMER}(A, ID\_CUSTOMER)$,

$I_{ORDERS.ID\_CUSTOMER}(A, ID\_CUSTOMER)$,

$I_{ORDER.TOTALPRICE}(A, TOTALPRICE)$.

Все индексы сортировались по значениям соответствующих атрибутов. Индексы $I_{CUSTOMER.ID\_CUSTOMER}$, $I_{ORDERS.ID\_CUSTOMER}$ фрагментировались и сегментировались на основе доменно-интервального принципа по домену [1, SF*630000], индекс $I_{ORDER.TOTALPRICE}$ фрагментировался и сегментировался транзитивно относительно индекса $I_{ORDERS.ID\_CUSTOMER}$. Все фрагментные и

сегментные интервалы, на которые делился домен, имели одинаковую длину. Сегменты индексов сжимались с помощью библиотеки Zlib [51, 111].

При выполнении запроса колоночный сопроцессор КСОП вычислял таблицу предварительных вычислений *P*(*A_ORDERS*, *A_CUSTOMER*) следующим образом:

$$P = \pi_{I_{CUSTOMER.ID\_CUSTOMER}.A \to A\_CUSTOMER,\ I_{ORDERS.ID\_CUSTOMER}.A \to A\_ORDERS} \Big(I_{CUSTOMER.ID\_CUSTOMER} \overset{\boxed{I_{CUSTOMER.ID\_CUSTOMER}.ID\_CUSTOMER = I_{ORDERS.ID\_CUSTOMER}.ID\_CUSTOMER}}{\bowtie} \left(I_{ORDERS.ID\_CUSTOMER} \bowtie \sigma_{TOTALPRICE \le Sel \cdot 100\,000} \left(I_{ORDERS.TOTALPRICE} \right) \right) \Big).$$

Вычисления фрагментов таблицы *P* производились параллельно на доступных процессорных узлах без обменов данными. Затем полученные фрагменты пересылались на узел с PostgreSQL, где *координатор* осуществлял их слияние в общую таблицу *P*. Вместо выполнения запроса Q1 в PostgreSQL выполнялся следующий запрос Q2 с использованием таблицы предварительных вычислений *P*:

```
# Запрос Q2
SELECT * FROM
        CUSTOMER INNER JOIN (
                P INNER JOIN ORDERS ON (ORDERS.A = P.A_ORDERS)
        ) ON (CUSTOMER.A=P.A_CUSTOMER);
```

При этом в PostgreSQL для атрибутов CUSTOMER.A и ORDERS.A предварительно были созданы кластеризованные индексы в виде B-деревьев.

4.2. Балансировка загрузки процессорных ядер Xeon Phi

В разделе 3.3 было сказано, что балансировку загрузки процессорных узлов, на которых располагаются распределенные колоночные индексы, можно осуществлять путем сдвигов границ фрагментных интервалов соответствующих доменов. В отличие от этого все сегментные интервалы для каждого конкретного домена имеют одинаковую длину. В случае, если значения атрибута в столбце распределены неравномерно, сегменты соответствующего колоночного индекса внутри одного процессорного узла могут значительно

отличаться по своим размерам. Потенциально это может привести к дисбалансу в загрузке процессорных ядер, так как каждый сегмент обрабатывается целиком на одном ядре. Однако, если число обрабатываемых сегментов значительно превышает количество процессорных ядер, дисбаланса загрузки не возникнет. Целью первого эксперимента было определить оптимальное соотношение между количеством сегментов и процессорных ядер в условиях перекоса в распределении данных. Использовались следующие параметры эксперимента:

– Вычислительная система: «Торнадо ЮУрГУ»;
– Количество задействованных узлов: 1;
– Используемый процессор: Xeon Phi;
– Количество задействованных ядер: 60;
– Количество нитей на ядро: 1;
– Коэффициент масштабирования базы данных: SF = 1;
– Коэффициент селективности: Sel=0.0005;
– Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform, «45-20», «65-20», 80-20»;
– Операция: построение таблицы предварительных вычислений *P*.

Результаты эксперимента представлены на рис. 13. Из графиков видно, что при малом количестве сегментов сильный перекос по данным приводит к существенному дисбалансу в загрузке процессорных ядер. В случае, когда количество сегментов равно 60 и совпадает с количеством ядер, время выполнения операции при распределении «80-20» более чем в четыре раза превышает время выполнения той же операции при равномерном (uniform) распределении. Однако при увеличении количества сегментов влияние перекоса по данным нивелируется. Для распределения «45-20» оптимальным оказывается число сегментов, равное 10 000, для распределения «65-20» – 20 000, и для распределения «80-20» – 200 000.

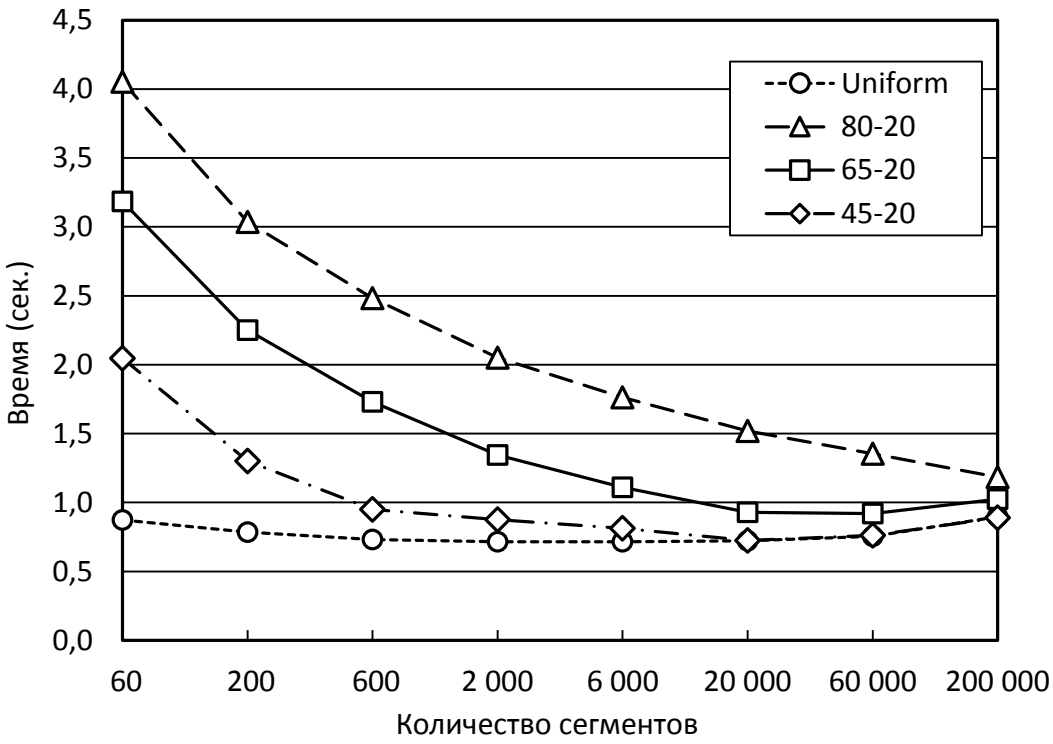


Рис. 13. Влияние количества сегментов на балансировку загрузки процессорных ядер сопроцессора Xeon Phi.

4.3. Влияние гиперпоточности

В большинстве случаев современные кластерные вычислительные системы оснащаются процессорами, поддерживающими технологию гиперпоточности (hyper-threading) [131], при включении которой каждое физическое ядро процессора определяется операционной системой как два или более логических ядра. У каждого логического ядра имеется свой набор регистров и контроллер прерываний. Остальные элементы физического ядра являются общими для всех логических ядер. При определенных рабочих нагрузках использование гиперпоточности позволяет увеличить производительность процессора.

ЦПУ Intel Xeon X5680 имеет аппаратную поддержку двух потоков на ядро, сопроцессор Intel Xeon Phi поддерживает четыре потока на ядро. Целью

Табл. 3. Время вычисления ТПВ при различном количестве потоков (сек.).

ПУ	Количество потоков (%)											
	8	17	25	33	42	50	58	67	75	83	92	100
CPU	2.25	1.14	0.78	0.59	0.47	0.39	0.37	0.34	0.32	0.31	0.29	0.28
MIC	2.39	1.22	0.85	0.73	0.65	0.59	0.61	0.64	0.68	0.69	0.73	0.79

второго эксперимента было определить, насколько эффективно гиперпоточность может быть применена при работе колоночного сопроцессора КСОП. Использовались следующие параметры эксперимента:

– Вычислительная система: «Торнадо ЮУрГУ»;
– Количество задействованных узлов: 1;
– Используемые процессорные устройства (ПУ):
  2 × Intel Xeon X5680 (CPU) / Intel Xeon Phi (MIC);
– Количество задействованных ядер: 12 / 60;
– Коэффициент масштабирования базы данных: SF = 1;
– Коэффициент селективности: Sel=0.0005;
– Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform;
– Операция: построение таблицы предварительных вычислений *P*.

Эксперимент сначала проводился на 2 × Intel Xeon X5680 (CPU), а затем на Intel Xeon Phi (MIC). Варьировался процент задействованных потоков. Для 2 × Intel Xeon X5680 величина 100% соответствует 24 потокам. Для Intel Xeon Phi величина 100% соответствует 240 потокам. Ускорение вычислялось по формуле $t_{x\%}/t_{20\%}$, где $t_{x\%}$ – время выполнения операции по вычислению ТПВ на *x*% потоков, $t_{20\%}$ – на 20% потоков. Результаты эксперимента представлены в табл. 3 и на рис. 14. Эксперимент показал, что для CPU производительность растет вплоть до максимального числа аппаратно поддерживаемых потоков. Однако, при использовании одного потока на ядро на

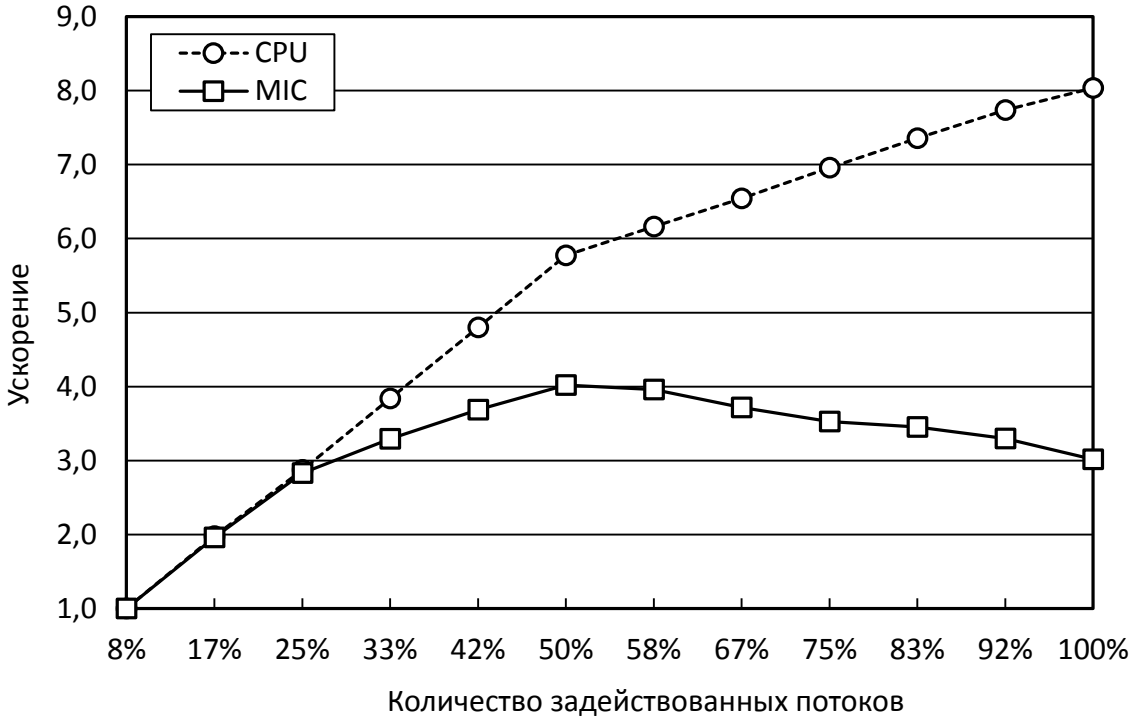


Рис. 14. Влияние гиперпоточности на ускорение.

CPU наблюдается ускорение, близкое к идеальному, а при использовании двух потоков на ядро прирост ускорения становиться более медленным. Применительно к MIC картина меняется. При использовании одного потока на ядро на MIC наблюдается ускорение, близкое к идеальному. При использовании двух потоков на ядро прирост ускорения становится более медленным. Использование же большего количества потоков на одно ядро ведет к деградации производительности.

4.4. Масштабируемость КСОП

В данном разделе исследуется масштабируемость колоночного сопроцессора КСОП на двух различных вычислительных системах и на базах данных двух разных масштабов. Масштабируемость является одной из важнейших характеристик при разработке параллельных СУБД для вычислительных систем с массовым параллелизмом. Основной численной характеристикой масштабируемости является ускорение, которое вычисляется по формуле t_x/t_k, где

t_k - время выполнения запроса на некоторой базовой конфигурации, включающей *k* процессорных узлов, t_x – время выполнения запроса на конфигурации, включающей *x* процессорных узлов при $x \geq k$. При использовании колоночного сопроцессора КСОП общее время выполнения запроса вычисляется по следующей формуле:

$$t = t_{openMP} + t_{MPI} + t_{merge} + t_{SQL},$$

где t_{openMP} – максимальное время вычисления фрагментов ТПВ и их сжатие на отдельных процессорных узлах; t_{MPI} – время пересылки сжатых фрагментов ТПВ на узел-координатор; t_{merge} – время их распаковки и слияния в единую таблицу; t_{SQL} – время выполнения запроса с использованием ТПВ на SQL-сервере. В описываемой реализации роль SQL-сервера играла СУБД PostgeSQL, установленная на узле-координаторе. Поэтому время t_{SQL} не зависело от количества используемых процессорных узлов. Это время будет исследовано в разделе 4.5. В данном разделе мы исследуем время, вычисляемое по формуле:

$$t_{total} = t_{openMP} + t_{MPI} + t_{merge}.$$

Первый эксперимент на масштабируемость проводился на кластере «Торнадо ЮУрГУ» и имел следующие параметры:

– Вычислительная система: «Торнадо ЮУрГУ»;
– Количество задействованных узлов: 60 – 210;
– Используемые процессорные устройства (ПУ):

  2 × Intel Xeon X5680;

– Использование гиперпоточности: не используется;
– Коэффициент масштабирования базы данных: SF = 1;
– Коэффициент селективности: 0.05 – 0.0005;

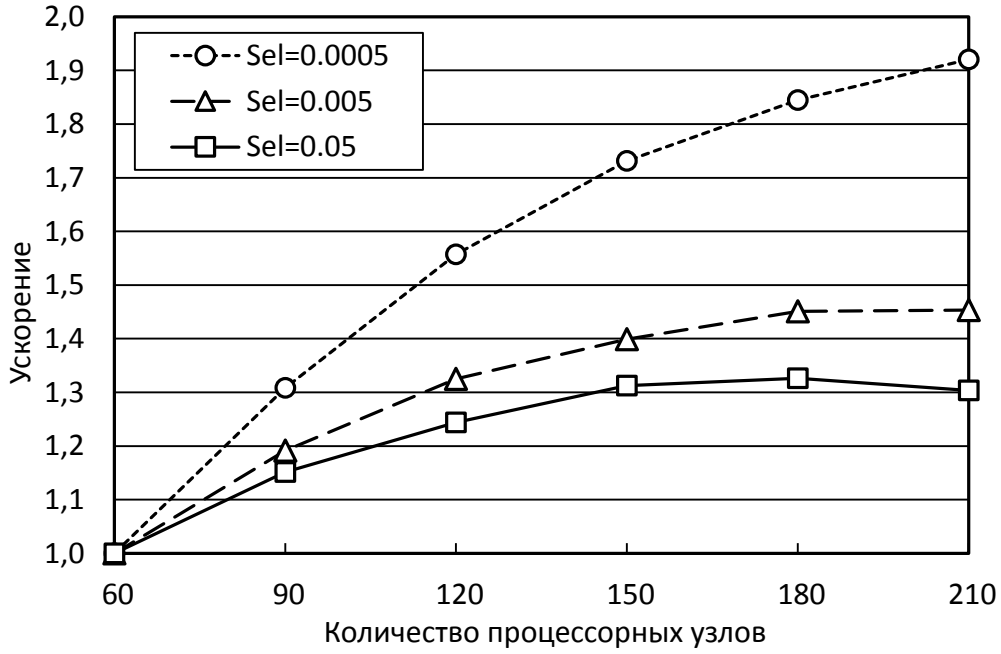


Рис. 15. Ускорение вычисления ТПВ на кластере «Торнадо ЮУрГУ» при SF = 1.

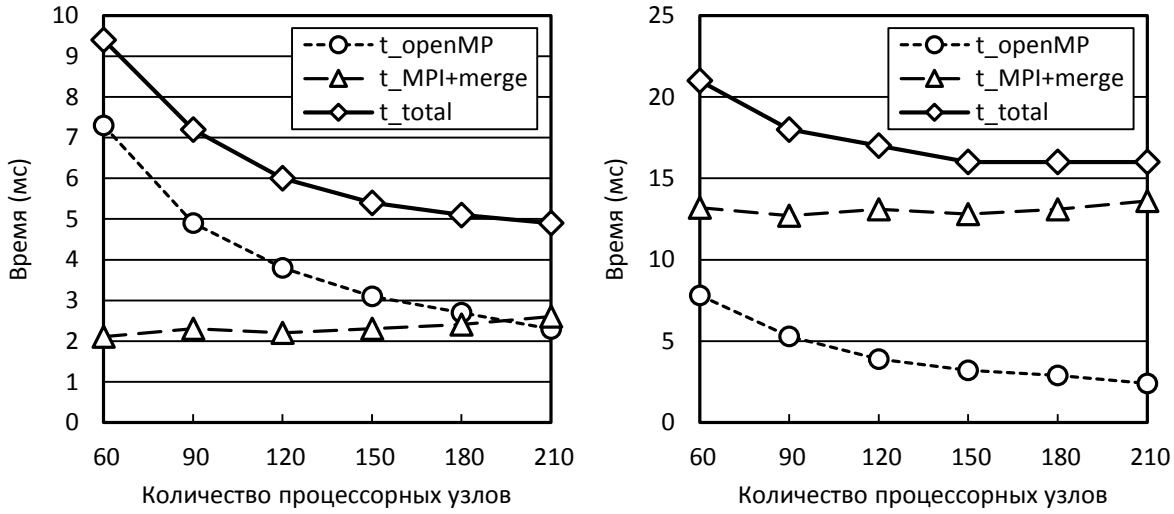
– Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform;
– Операция: построение таблицы предварительных вычислений *P*.

Результаты эксперимента приведены на рис. 15 и в табл. 4. Графики ускорения вычисления ТПВ на рис. 15 показывают, что селективность запроса *Sel* оказывается фактором, ограничивающим масштабируемость. Так, при *Sel* = 0.05 масштабируемость ограничена 150 вычислительными узлами, при *Sel* = 0.005 – 180, а при *Sel* = 0.0005 превышает 210. Это связано с тем, что при увеличении селективности увеличивается размер ТПВ. При *Sel* = 0.0005 он составляет порядка 31 500 кортежей, при *Sel* = 0.005 – 315 000, а при *Sel* = 0.05 – 3 150 000 (значения даны для масштабного коэффициента SF = 1). На рис. 16 изображены зависимости выполнения времени подопераций от количества процессорных узлов. При малой селективности (рис. 16, а) сумма времени пересылки фрагментов ТПВ, их сжатия, распаковки и слияния (t_MPI+merge) остается практически неизменной при увеличении процессорных узлов. Время вычисления фрагментов ТПВ на процессорных

Табл. 4. Время вычисления ТПВ на кластере «Торнадо ЮУрГУ» при SF = 1.

Время (сек.)	Количество процессорных узлов					
	60	90	120	150	180	210
Sel = 0.05						
t_{openMP}	0.0078	0.0053	0.0039	0.0032	0.0029	0.0024
t_{MPI}	0.0068	0.0083	0.0087	0.0087	0.0092	0.0096
t_{merge}	0.0064	0.0044	0.0044	0.0041	0.0039	0.0040
t_{total}	**0.021**	**0.018**	**0.017**	**0.016**	**0.016**	**0.016**
Sel = 0.005						
t_{openMP}	0.0074	0.0049	0.0038	0.0031	0.0027	0.0023
t_{MPI}	0.0032	0.0032	0.0034	0.0037	0.0043	0.0043
t_{merge}	0.00025	0.00099	0.00099	0.00088	0.00052	0.00093
t_{total}	**0.0108**	**0.0091**	**0.0082**	**0.0077**	**0.0075**	**0.0075**
Sel = 0.0005						
t_{openMP}	0.0073	0.0049	0.0038	0.0031	0.0027	0.0023
t_{MPI}	0.0021	0.0023	0.0022	0.0023	0.0024	0.0026
t_{merge}	0.000006	0.000006	0.000006	0.000008	0.000005	0.000005
t_{total}	**0.0094**	**0.0072**	**0.0060**	**0.0054**	**0.0051**	**0.0049**

узлах (t_openMP) с ростом количества узлов уменьшается, и вплоть то 210 узлов превышает t_MPI+merge. Поэтому общее время (t_total) также уменьшается, что обеспечивает положительное ускорение. Однако при большой



а) Sel = 0.0005 б) Sel = 0.05

Рис. 16. Время выполнения подопераций при вычислении ТПВ на «Торнадо ЮУрГУ» при SF = 1.

селективности (рис. 16, б) t_MPI+merge, оставаясь практически неизменным, значительно превышает t_openMP, что препятствует существенному уменьшению общего времени при увеличении количества процессорных узлов.

При увеличение коэффициента масштабируемости базы данных SF до значения 10 картина существенно меняется (см. рис. 17 и табл. 5). На рис. 17 видно, что в этом случае кривая ускорения для *Sel* = 0.0005 становится практически линейной, а для *Sel* = 0.005 приближается к линейной. Однако, и в этом случае при большом значении коэффициента селективности *Sel* = 0.05 масштабируемость ограничивается 150 процессорными узлами. Как видно из табл. 5, причина та же самая: при *Sel* = 0.05 время передачи, распаковки и слияния ТПВ ($t_{MPI} + t_{merge}$) меняется мало и существенно превышает время ее вычисления t_{openMP}, в то время как при *Sel* = 0.0005 значение ($t_{MPI} + t_{merge}$) почти на порядок меньше t_{openMP}. В соответствие с этим можно сделать вывод, что при малой селективности запроса КСОП демонстрирует на больших базах данных ускорение, близкое к линейному.

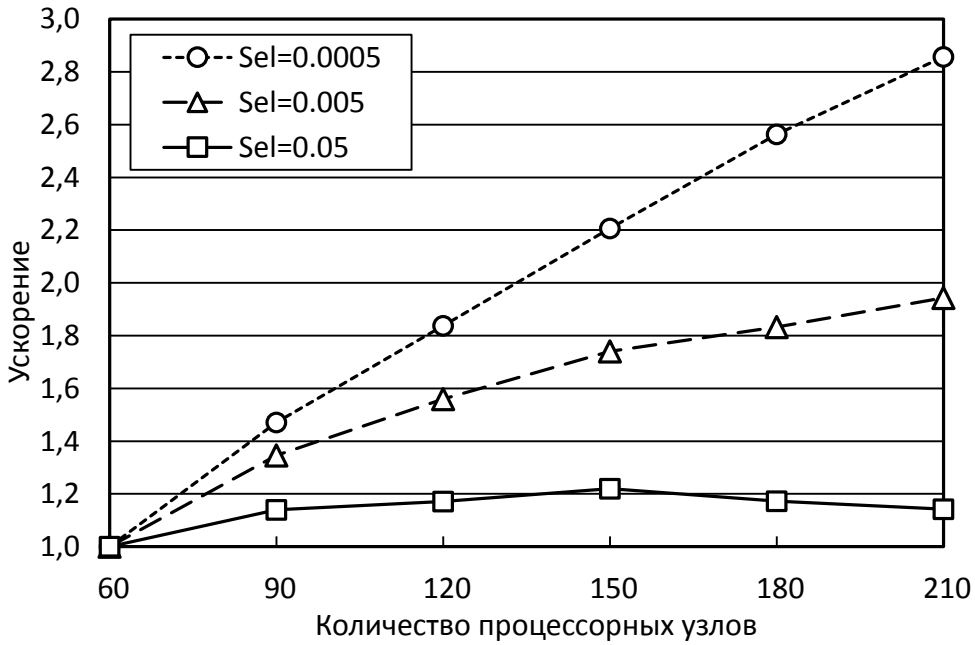


Рис. 17. Ускорение вычисления ТПВ на кластере «Торнадо ЮУрГУ» при SF = 10.

Табл. 5. Время вычисления ТПВ на кластере «Торнадо ЮУрГУ» при SF = 10.

Время (сек.)	Количество процессорных узлов					
	60	90	120	150	180	210
Sel = 0.05						
t_{openMP}	0.071	0.048	0.036	0.029	0.024	0.02
t_{MPI}	0.065	0.061	0.059	0.056	0.065	0.076c
t_{merge}	0.104	0.101	0.105	0.105	0.111	0.114
t_{total}	**0.24**	**0.21**	**0.20**	**0.19**	**0.20**	**0.21**
Sel = 0.005						
t_{openMP}	0.064	0.043	0.033	0.026	0.022	0.019
t_{MPI}	0.013	0.013	0.011	0.013	0.013	0.014
t_{merge}	0.0063	0.0063	0.0095	0.0094	0.0097	0.01
t_{total}	**0.083**	**0.062**	**0.053**	**0.048**	**0.045**	**0.043**
Sel = 0.0005						
t_{openMP}	0.063	0.043	0.032	0.026	0.022	0.019
t_{MPI}	0.0078	0.0048	0.007	0.0060	0.006	0.006
t_{merge}	0.0002	0.00019	0.00002	0.00002	0.00002	0.00002
t_{total}	**0.071**	**0.048**	**0.039**	**0.032**	**0.028**	**0.025**

При исследовании масштабируемости колоночного сопроцессора КСОП важным вопросом является ее зависимость от аппаратной архитектуры вычислительной системы. Как было сказано в главе 3, КСОП реализован с использованием аппаратно независимых параллельных технологий MPI и OpenMP. Поэтому он может работать как на ЦПУ Intel Xeon X5680, так и на сопроцессоре Intel Xeon Phi. Масштабируемость КСОП на вычислительном кластере «RSC PetaStream» была исследована в эксперименте, который имел следующие параметры:

– Вычислительная система: «RSC PetaStream»;
– Количество задействованных узлов: 10 – 60;
– Используемые процессорные устройства (ПУ): Intel Xeon Phi 7120;
– Использование гиперпоточности: не используется;
– Коэффициент масштабирования базы данных: SF = 1;
– Коэффициент селективности: 0.5 – 0.0005;
– Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform;
– Операция: построение таблицы предварительных вычислений *P*.

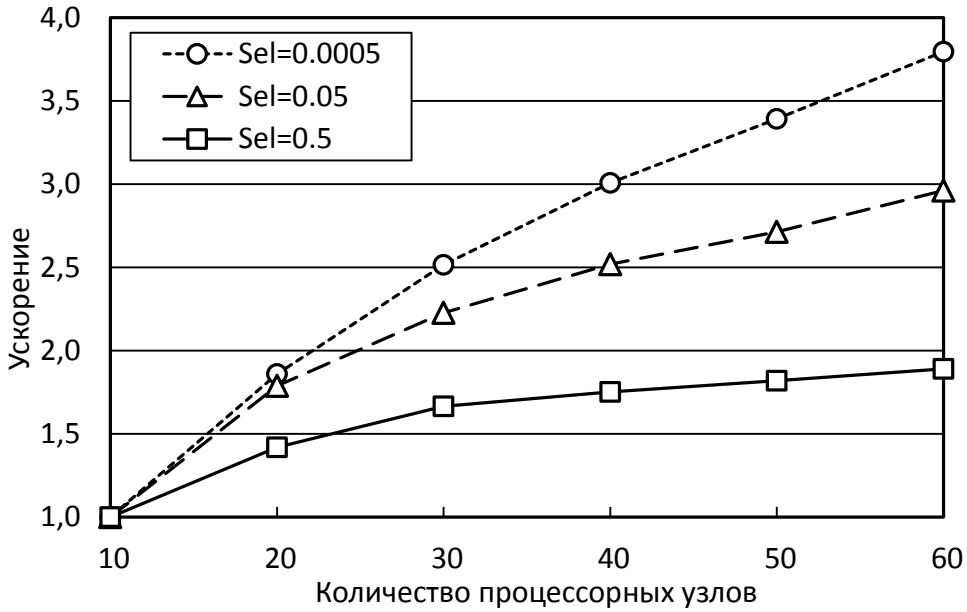


Рис. 18. Ускорение вычисления ТПВ на кластере «RSC PetaStream» при SF = 1.

Табл. 6. Время вычисления ТПВ на кластере «RSC PetaStream» при SF = 1.

Время (сек.)	Количество процессорных узлов					
	10	20	30	40	50	60
Sel = 0.5						
t_{openMP}	0.076	0.039	0.027	0.021	0.018	0.014
t_{MPI}	0.020	0.017	0.015	0.015	0.015	0.016
t_{merge}	0.056	0.051	0.049	0.050	0.050	0.050
t_{total}	**0.152**	**0.107**	**0.091**	**0.086**	**0.083**	**0.080**
Sel = 0.05						
t_{openMP}	0.074	0.037	0.026	0.021	0.017	0.014
t_{MPI}	0.0050	0.0056	0.0072	0.0066	0.0078	0.0089
t_{merge}	0.0060	0.0054	0.0048	0.0064	0.0062	0.0061
t_{total}	**0.085**	**0.048**	**0.038**	**0.034**	**0.031**	**0.029**
Sel = 0.005						
t_{openMP}	0.074	0.037	0.026	0.020	0.017	0.014
t_{MPI}	0.0040	0.0048	0.0053	0.0058	0.0057	0.0070
t_{merge}	0.00097	0.00125	0.00072	0.0022	0.0033	0.0030
t_{total}	**0.079**	**0.043**	**0.032**	**0.028**	**0.026**	**0.024**
Sel = 0.0005						
t_{openMP}	0.074	0.037	0.026	0.020	0.017	0.014
t_{MPI}	0.0025	0.0048	0.0049	0.0059	0.0059	0.0059
t_{merge}	0.00055	0.00017	0.00014	0.00010	0.00009	0.00008
t_{total}	**0.077**	**0.042**	**0.031**	**0.026**	**0.023**	**0.020**

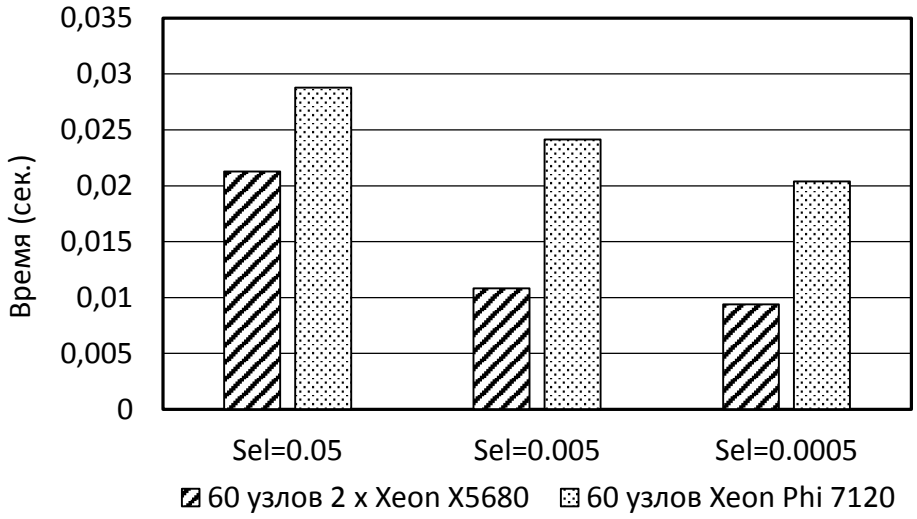


Рис. 19. Сравнение времени вычисления ТПВ на кластерах «Торнадо ЮУрГУ» (60 узлов 2 × Xeon X5680) и «RSC PetaStream» (60 узлов Xeon Phi 7120) при SF = 1.

Результаты эксперимента представлены на рис. 18 и в табл. 6. Они показывают, что масштабируемость КСОП и в этом случае приближается к линейной при уменьшении селективности запроса.

Используя данные из табл. 4 и табл. 6, было проведено сравнение производительности вычислительных систем «Торнадо ЮУрГУ» (60 узлов 2 × Xeon X5680) и «RSC PetaStream» (60 узлов Xeon Phi 7120). Результаты, приведенные на рис. 19, показывают что во всех случаях 60-ти узловая конфигурация «Торнадо ЮУрГУ» на базе 2 × Xeon X5680 превосходит по производительности аналогичную конфигурацию «RSC PetaStream» на базе Xeon Phi 7120. При этом следует отметить, что при разработке КСОП не производилась никакая специальная оптимизация программы под архитектуру MIC. Такая настройка может существенным образом повысить производительность системы [78].

4.5. Использование КСОП при выполнении SQL-запросов

В заключительном эксперименте было исследовано в какой мере использование колоночного сопроцессора КСОП может ускорить выполнение запроса

класса OLAP в реляционной СУБД. Эксперимент проводился при следующих параметрах:

– Вычислительная система: «Торнадо ЮУрГУ»;
– Конфигурация PostgreSQL: 1 процессорный узел с SSD Intel;
– Конфигурация КСОП: 210 процессорных узлов;
– Используемые процессорные устройства (ПУ):

  2 × Intel Xeon X5680;

– Использование гиперпоточности: не используется;
– Коэффициент масштабирования базы данных: SF = 1;
– Коэффициент селективности: 0.05 – 0.0005;
– Распределение значений в колонке ORDERS.ID_CUSTOMER: uniform;
– Операция: выполнение SQL-запроса.

В ходе выполнения эксперимента исследовались следующие три конфигурации (см. табл. 7):

1) PostgreSQL: выполнение запроса Q1 (см. стр. 105) без создания индексных файлов в виде B-деревьев;
2) PostgreSQL & B-Trees: выполнение запроса Q1 (см. стр. 105) с предварительным созданием индексных файлов в виде B-деревьев для атрибутов CUSTOMER.ID_CUSTOMER и ORDERS.ID_CUSTOMER;
3) PostgreSQL & CCOP: выполнение запроса Q2 (см. стр. 106) с использованием ТПВ *P* и предварительным созданием индексных файлов в виде B-деревьев для атрибутов CUSTOMER.A и ORDERS.A.

В последнем случае ко времени выполнения запроса добавлялось время создания ТПВ колоночным сопроцессором КСОП (CCOP). В каждом случае замерялось время первого и повторного запуска запроса. Это связано

Табл. 7. Время вычисления SQL-запроса и ускорение в сравнении с PostgreSQL при SF=1.

Конфигурация	Время в минутах					
	Sel = 0.0005		Sel = 0.005		Sel = 0.05	
	1-й запуск	2-й запуск	1-й запуск	2-й запуск	1-й запуск	2-й запуск
PostgreSQL	7.3	1.21	7.6	1.29	7.6	1.57
PostgreSQL & B-Trees	2.62	2.34	2.83	2.51	2.83	2.63
PostgreSQL & CCOP	0.073	0.008	0.65	0.05	2.03	1.72
Ускорение						
$\frac{t_{PostgreSQL}}{t_{PostgreSQL\ \&\ CCOP}}$	100	151	12	27	4	0.9
$\frac{t_{PostgreSQL\&B-Trees}}{t_{PostgreSQL\ \&\ CCOP}}$	36	293	4	50	1.4	1.53

с тем, что после первого выполнения запроса PostgreSQL собирает статистическую информацию, сохраняемую в словаре базы данных, которая затем используется для оптимизации плана выполнения запроса. Эксперименты показали, что при отсутствии индексов в виде B-деревьев использование колоночного сопроцессора позволяет увеличить производительность выполнения запроса в 100 – 150 раз для коэффициента селективности Sel = 0.0005. Однако при больших значениях коэффициента селективности эффективность использования КСОП может снижаться вплоть до отрицательных значений (ускорение меньше единицы).

4.6. Выводы по главе 4

В этой главе были описаны эксперименты над синтетической базой данных большого размера, исследующие эффективность описанного в главе 3 колоночного сопроцессора КСОП. Эксперименты показали, что подходы и методы параллельного выполнения запросов класса OLAP, разработанные в главе 2 на базе доменно-колоночной модели, демонстрируют хорошую масштабируемость (до нескольких сотен процессорных узлов и десятков тысяч

процессорных ядер) для запросов с большой селективностью, которые являются типичными для хранилищ данных. Использование колоночного сопроцессора КСОП во взаимодействии с СУБД PostgreSQL позволило повысить эффективность выполнения таких запросов более чем на два порядка. Однако, эффективность использования КСОП снижается при уменьшении размеров базы данных и при увеличении размеров результирующего отношения. Результаты, описанные в этой главе, опубликованы в работах [2, 4, 5, 8, 76].

ЗАКЛЮЧЕНИЕ

В диссертационной работе были рассмотрены вопросы разработки и исследования эффективных методов параллельной обработки сверхбольших баз данных с использованием колоночного представления информации, ориентированных на кластерные вычислительные системы, оснащенные многоядерными ускорителями, и допускающих интеграцию с реляционными СУБД. Введены колоночные индексы с суррогатными ключами. Предложена доменно-колоночная модель распределения данных по узлам многопроцессорной системы, на основе которой разработаны формальные методы декомпозиции реляционных операций на части, которые могут выполняться независимо (без обменов данными) на различных вычислительных узлах и процессорных ядрах. Корректность методов декомпозиции во всех случаях подтверждена математическими доказательствами. Разработанные методы реализованы в колоночном сопроцессоре КСОП, который работает на кластерных вычислительных системах, в том числе оснащенных многоядерными ускорителями, и может применяться в сочетании с реляционной СУБД для выполнения ресурсоемких операций. Взаимодействие СУБД с КСОП осуществляется через специальный драйвер, который устанавливается на SQL-сервере, где работает СУБД. Для организации этого взаимодействия в код СУБД встраивается специальный коннектор.

Основные результаты, полученные в ходе выполнения диссертационного исследования являются новыми и не покрываются ранее опубликованными научными работами других авторов, обзор которых был дан в разделе 1.4. Отметим основные отличия.

В работе [84] описываются индексы колоночной памяти (column store indexes), внедренные в Microsoft SQL Server 11. Этот подход фактически предполагает создание двух независимых исполнителя запросов (один для строчного хранилища, другой для колоночного) в рамках одной СУБД. Под-

ход, описанный в диссертации, предполагает реализацию колоночных индексов в рамках отдельной программной системы – колоночного сопроцессора КСОП. Кроме этого, в работе в работе [84] отсутствует описание методов фрагментации индексов колоночной памяти и ничего не говорится о методах распараллеливания операций над индексами колоночной памяти. Также следует отметить, что индексы колоночной памяти представляются в Microsoft SQL Server 11 в виде BLOB-объектов, хранимых на дисках. В КСОП колоночные индексы в распределенном виде хранятся в оперативной памяти кластерной вычислительной системы.

Методы, предложенные в работе [52] для включения в строчную СУБД колоночно-ориентированной обработки запросов, требуют глубокой модернизации СУБД и не приспособлены для работы на кластерных вычислительных системах с распределенной памятью. В отличие от этого КСОП требует минимальной модификации СУБД путем добавления в нее специального коннектора, и показывает масштабируемость близкую к линейной на кластерных вычислительных системах с сотнями процессорных узлов и десятками тысяч процессорных ядер.

В работе [43] в строчной СУБД вводятся дополнительные структуры данных, называемые c-таблицами, напоминающие колоночные индексы КСОП, однако полностью отсутствуют аспекты, связанные распределением данных и параллельной обработкой.

Подход к эмуляции колоночного хранилища в строчной СУБД, основанный на материализованных представлениях [24], не позволяет соединять в одном представлении колонки разных таблиц, что ограничивает его применимость для параллельного выполнения соединений в многопроцессорных системах с распределенной памятью. В отличие от этого КСОП способен выполнять соединения без обменов данными на кластерных вычислительных системах с большим количеством процессорных узлов.

Схема зеркалирования данных «*разбитое зеркало*» (*fractured mirror*) из [109] позволяет организовывать эффективную параллельную обработку фрагментно-независимых запросов в колоночном стиле на кластерной вычислительной системе. Однако, если запрос зависит от способа фрагментации данных, происходит деградация производительности системы из-за большого количества пересылок данных между процессорными узлами. Доменно-интервальная фрагментация, используемая в КСОП, позволяет избежать таких пересылок.

Другие подходы к эмуляции колоночного хранилища в строчной СУБД, описанные в [24], здесь не рассматриваются, так как они показываю худшую производительность по сравнению с обычными реляционными СУБД. В отличие от них КСОП позволяет ускорить выполнение запросов к хранилищу данных в сотни раз по сравнению с реляционной СУБД PostgreSQL.

Результаты, полученные в ходе настоящего диссертационного исследования, могут применяться при создании масштабируемых колоночных сопроцессоров для существующих коммерческих и свободно-распространяемых SQL-СУБД. Это позволит обрабатывать сверхбольшие хранилища данных на кластерных вычислительных системах, в том числе с узлами, включающими в себя многоядерные ускорители типа GPU или MIC.

В качестве направлений дальнейших исследований можно выделить следующие.

– Разработка и исследование методов интеграции КСОП со свободно распространяемыми реляционными СУБД типа PostgreSQL.
– Интеграция в КСОП легковесных методов сжатия, не требующих распаковки для выполнения операций над ними.
– Обобщение подходов и методов, предложенных в диссертации, на многомерные данные.

Работа выполнена при финансовой поддержке Минобрнауки РФ в рамках ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2014—2020 годы» (Госконтракт № 14.574.21.0035).

ЛИТЕРАТУРА

1. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс. М.: Издательский дом «Вильямс», 2004. 1088 с.
2. Иванова Е.В. Исследование эффективности использования фрагментированных колоночных индексов при выполнении операции естественного соединения с использованием многоядерных ускорителей // Параллельные вычислительные технологии (ПаВТ'2015): труды международной научной конференции (30 марта - 3 апреля 2015 г., г. Екатеринбург). Челябинск: Издательский центр ЮУрГУ, 2015. С. 393-398.
3. Иванова Е.В. Использование распределенных колоночных хеш-индексов для обработки запросов к сверхбольшим базам данных // Научный сервис в сети Интернет: многообразие суперкомпьютерных миров: Труды Международной суперкомпьютерной конференции (22-27 сентября 2014 г., Новороссийск). М.: Изд-во МГУ, 2014. С. 102-104.
4. Иванова Е.В. Соколинский Л.Б. Колоночный сопроцессор баз данных для кластерных вычислительных систем // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 4. С. 5-31.
5. Иванова Е.В. Соколинский Л.Б. Использование сопроцессоров Intel Xeon Phi для выполнения естественного соединения над сжатыми данными // Вычислительные методы и программирование: Новые вычислительные технологии. 2015. Т. 16. Вып. 4. С. 534-542.
6. Иванова Е.В. Соколинский Л.Б. Параллельная декомпозиция реляционных операций на основе распределенных колоночных индексов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 4. С. 80-100.

7. Иванова Е.В. Соколинский Л.Б. Декомпозиция операций пересечения и соединения на основе доменно-интервальной фрагментации колоночных индексов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2015. Т. 4, № 1. С. 44-56.
8. Иванова Е.В., Соколинский Л.Б. Использование сопроцессоров Intel Xeon Phi для выполнения естественного соединения над сжатыми данными // Суперкомпьютерные дни в России: Труды международной конференции (28-29 сентября 2015 г., г. Москва). М.: Изд-во МГУ, 2015. С. 190-198.
9. Иванова Е.В. Соколинский Л.Б. Использование распределенных колоночных индексов для выполнения запросов к сверхбольшим базам данных // Параллельные вычислительные технологии (ПаВТ'2014): труды международной научной конференции (1–3 апреля 2014 г., г. Ростов-на-Дону). Челябинск: Издательский центр ЮУрГУ, 2014. С. 270-275.
10. Иванова Е.В., Соколинский Л.Б. Декомпозиция операции группировки на базе распределенных колоночных индексов // Наука ЮУрГУ. Челябинск: Издательский центр ЮУрГУ, 2015. С. 15-23.
11. Корнеев В. Следующее поколение суперкомпьютеров // Открытые системы. 2008. № 8. С. 14-19.
12. Костенецкий П.С., Соколинский Л.Б. Моделирование иерархических многопроцессорных систем баз данных // Программирование. 2013. Т. 39, № 1. С. 3-22.
13. Кузнецов С.Д. SQL. Язык реляционных баз данных. М.: Майор, 2001. 192 с.
14. Лепихов А.В., Соколинский Л.Б. Обработка запросов в СУБД для кластерных систем // Программирование. 2010. № 4. С. 25-39.

15. Массивно-параллельный суперкомпьютер RSC PetaStream. URL: http://rscgroup.ru/ru/our-solutions/massivno-parallelnyy-superkompyuter-rsc-petastream (дата обращения: 02.10.2015).
16. Соколинский Л.Б. Обзор архитектур параллельных систем баз данных // Программирование. 2004. № 6. С. 49-63.
17. Соколинский Л.Б. Параллельные системы баз данных. М.: Издательство Московского государственного университета, 2013. 184 с.
18. Суперкомпьютер «Торнадо ЮУрГУ». URL: http://supercomputer.susu.ru/computers/tornado (дата обращения: 02.10.2015).
19. Чернышев Г. А. Организация физического уровня колоночных СУБД // Труды СПИИРАН. 2013. №7. Вып. 30. С. 204-222.
20. A Drill-Down into EXASolution. Technical Whitepaper. EXASOL AG, 2014. 15 p. URL: http://info.exasol.com/whitepaper-exasolution-2-en.html (дата обращения: 22.10.2015).
21. A Peek under the Hood. Technical Whitepaper. EXASOL AG, 2014. 16 p. URL: http://www.breos.com/sites/default/files/pdf/downloads/exasol_whitepaper.pdf (дата обращения: 22.10.2015).
22. Abadi D.J., Boncz P.A., Harizopoulos S. Column-oriented Database Systems // Proceedings of the VLDB Endowment. 2009. Vol. 2, No. 2. P. 1664-1665.
23. Abadi D.J., Boncz P.A., Harizopoulos S., Idreos S., Madden S. The Design and Implementation of Modern Column-Oriented Database Systems // Foundations and Trends in Databases. 2013. Vol. 5, No. 3. P. 197-280.
24. Abadi D.J., Madden S.R., Hachem N. Column-Stores vs. Row-Stores: How Different Are They Really? // Proceedings of the 2008 ACM SIGMOD international conference on Management of data, June 9-12, 2008, Vancouver, BC, Canada. ACM, 2008. P. 967-980.
25. Abadi D. J., Madden S. R., Ferreira M. Integrating compression and execution in column-oriented database systems // Proceedings of the 2006 ACM

SIGMOD international conference on Management of data, June 26-29, 2006, Chicago, Illinois. ACM, 2006. P. 671-682.

26. Abadi D. J., Myers D. S., DeWitt D. J., Madden S. R. Materialization strategies in a column-oriented DBMS // Proceedings of the 23rd International Conference on Data Engineering (ICDE), April 15-20, 2007, Istanbul, Turkey. IEEE, 2007. P. 466-475.
27. Aghav S. Database compression techniques for performance optimization // Proceedings of the 2010 2nd International Conference on Computer Engineering and Technology (ICCET), April 16-18, 2010, Chengdu Convention Center Chengdu, China. IEEE, 2010. P. 714-717.
28. Actian SQL Analytics in Hadoop. A Technical Overview. Actian Corporation, 2015. 16 p. URL: http://bigdata.actian.com/SQLAnalyticsinHadoop (дата обращения 27.10.2015).
29. Architecture of SQLite. URL: http://www.sqlite.org/arch.html (дата обращения 11.10.2015)
30. Bakkum P., Chakradhar S. Efficient data management for GPU databases. URL: http://pbbakkum.com/virginian/paper.pdf (дата обращения 11.09.2015)
31. Barber R., Bendel P., Czech M., Draese O., Ho F., Hrle N., Idreos S., Kim M.-S., Koeth O., Lee J.-G., Li T.T., Lohman G. M., Morfonios K., Müller R., Murthy K., Pandis I., Qiao L., Raman V., Sidle R., Stolze K., Szabo S. Business Analytics in (a) Blink // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 9-14.
32. Bassiouni M. A. Data Compression in Scientific and Statistical Databases // IEEE Transactions on Software Engineering. 1985. Vol. 11, No. 10. P. 1047-1058.
33. Batory D. S. On searching transposed files // ACM Transactions on Database Systems. 1979. Vol. 4, No. 4. P. 531-544.

34. Boncz P.A., Kersten M.L. MIL primitives for querying a fragmented world // VLDB Journal, 1999. Vol. 8, No. 2. P. 101-119.
35. Boncz P.A., Kersten M.L., Manegold S. Breaking the memory wall in MonetDB // Communications of the ACM, 2008. Vol. 51, No. 12. P. 77-85.
36. Boncz P.A., Zukowski M., Nes N. MonetDB/X100: Hyper-pipelining query execution // Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR), January 4-7, Asilomar, CA, USA. 2005. P. 225-237.
37. Breß S. Why it is time for a HyPE: a hybrid query processing engine for efficient GPU coprocessing in DBMS // Proceedings of the VLDB Endowment. 2013. Vol. 6, No. 12. P. 1398-1403.
38. Breß S., Beier F., Rauhe H., et al. Efficient Co-Processor Utilization in Database Query Processing // Information Systems, 2013. Vol. 38, No. 8. P. 1084-1096.
39. Breß S., Geist I., Schallehn E., Mory M., Saake G. A framework for cost based optimization of hybrid CPU/GPU query plans in database systems // Control and Cybernetics. 2012. Vol. 41, No. 4. P. 715-742.
40. Breß S., Heimel M., Siegmund N., Bellatreche L., Saake G. GPU-Accelerated Database Systems: Survey and Open Challenges // Transactions on Large-Scale Data- and Knowledge-Centered Systems. 2014. Vol. 15. P. 1-35.
41. Breß S., Siegmund N., Bellatreche L., Saake G. An operator-stream-based scheduling engine for effective GPU coprocessing // Proceedings of the 17th East European Conference on Advances in Databases and Information Systems (ADBIS 2013), September 1-4, 2013, Genoa, Italy. Springer, 2013. P. 288-301.
42. Broneske D., Breß S., Heimel M., Saake G. Toward hardware-sensitive database operations // Proceedings of the 17th International Conference on

Extending Database Technology (EDBT 2014), March 24-28, 2014, Athens, Greece. OpenProceedings.org, 2014. P. 229-234.

43. Bruno N. Teaching an Old Elephant New Tricks // Online Proceedings of Fourth Biennial Conference on Innovative Data Systems Research (CIDR 2009), Asilomar, CA, USA, January 4-7, 2009. www.cidrdb.org, 2009. URL: http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_2.pdf (дата обращения 19.09.2015).
44. Cell Broadband Engine Architecture. IBM. URL: https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA/$file/CBEA_v1.02_11Oct2007_pub.pdf (дата обращения 04.09.2015).
45. Chaudhuri S., Dayal U. An Overview of Data Warehousing and OLAP Technology // SIGMOD Record, 1997. Vol. 26, No. 1. P. 65-74.
46. Chen Z., Gehrke J., Korn F. Query Optimization in Compressed Database Systems // Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, May 20-24, 2011, Scottsdale, AZ, USA. ACM, 2001. P. 271-282.
47. Copeland G. P., Khoshafian S. N. A decomposition storage model // Proceedings of the 1985 ACM SIGMOD international conference on Management of data, May 28-31, 1985, Austin, Texas, USA. ACM Press, 1985. P. 268-279.
48. Dees J., Sanders P. Efficient many-core query execution in main memory column stores // Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE 2013), April 8-12, 2013, Brisbane, Australia. IEEE, 2013. P. 350-361.
49. Deshmukh P.A. Review on Main Memory Database // International Journal of Computer & Communication Technology, 2011. Vol. 2, No. 7. P. 54-58.
50. Deutsch P. DEFLATE Compressed Data Format Specification version 1.3. United States: RFC Editor, 1996. URL: https://www.ietf.org/rfc/rfc1951.txt (дата обращения: 16.11.2015).

51. Deutsch P., Gailly J.-L. ZLIB Compressed Data Format Specification version 3.3. United States: RFC Editor, 1996. URL: ZLIB Compressed Data Format (дата обращения: 16.11.2015).
52. El-Helw A., Ross K.A., Bhattacharjee B., Lang C.A., Mihaila G.A. Column-oriented query processing for row stores // Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP (DOLAP '11), October 28, 2011, Glasgow, United Kingdom. ACM, 2011. P. 67-74.
53. EXASolution. Business Whitepaper. EXASOL AG, 2015. 11 p. URL: http://info.exasol.com/business-whitepaper-exasolution-en.html (дата обращения: 27.10.2015).
54. Fang J., Varbanescu A.L., Sips H. Sesame: A User-Transparent Optimizing Framework for Many-Core Processors // Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid2013), May 13–16, 2013, Delft, Netherlands. IEEE, 2013. P. 70-73.
55. Fang R., He B., Lu M., Yang K., Govindaraju N. K., Luo Q., Sander P. V. GPUQP: query co-processing using graphics processors // Proceedings of the ACM SIGMOD International Conference on Management of Data, June 12-14, 2007, Beijing, China. ACM, 2007. P. 1061-1063.
56. Fang W., He B., Luo Q. Database compression on graphics processors // Proceedings of the VLDB Endowment. 2010. Vol. 3, No. 1-2. P. 670-680.
57. Färber F., May N., Lehner W., Große P., Müller I., Rauhe H., Dees J. The SAP HANA Database – An Architecture Overview // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 28-33.
58. Furtado P. A Survey of Parallel and Distributed Data Warehouses // International Journal of Data Warehousing and Mining, 2009. Vol. 5, No. 5. P. 57-77.

59. Garcia-Molina H., Salem K. Main Memory Database Systems: An Overview // IEEE Transactions On Knowledge and Data Engineering, 1992. Vol. 4, No. 6. P. 509-516.
60. Ghodsnia P. An in-GPU-memory column-oriented database for processing analytical workloads // The VLDB PhD Workshop. VLDB Endowment (2012). Vol. 1.
61. Global Enterprise Big Data Trends: 2013. Microsoft corp. survey, 2013. URL: https://news.microsoft.com/download/presskits/bigdata/docs/bigdata_021113.pdf (дата обращения: 19.10.2015)
62. Golfarelli M., Rizzi S. A Survey on Temporal Data Warehousing // International Journal of Data Warehousing and Mining, 2009. Vol. 5, No. 1. P. 1-17.
63. Graefe G. Encapsulation of parallelism in the Volcano query processing system // Proceedings of the 1990 ACM SIGMOD international conference on Management of data, May 23-25, 1990, Atlantic City, NJ. ACM, 1990. P. 102-111.
64. Gray J., Sundaresan P., Englert S., Baclawski K., Weinberger P. J. Quickly generating billion-record synthetic databases // Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, May 24-27, 1994, Minneapolis, Minnesota. ACM Press, 1994. P. 243-252.
65. Gregg C., Hazelwood K. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer // Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2011), April 10-12, 2011, Austin, TX, USA. IEEE, 2011. P. 134-144.
66. Gschwind M., The Cell Broadband Engine: Exploiting Multiple Levels of Parallelism in a Chip Multiprocessor // International Journal of Parallel Programming. 2007. Vol. 35, No. 3. P. 233-262.

67. Harizopoulos S., Abadi D., Madden S., Stonebraker M. OLTP Through the Looking Glass, And What We Found There // Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, June 10-12,Vancouver, BC, Canada. ACM, 2008. P. 981-992.
68. He B., Lu M., Yang K., Fang R., Govindaraju N.K., Luo Q., Sander P.V. Relational query co-processing on graphics processors // ACM Transactions on Database System. 2009. Vol. 34, No. 21. P. 1-35.
69. He B., Yang K., Fang R., Lu M., Govindaraju N., Luo Q., Sander P. Relational joins on graphics processors // Proceedings of the ACM SIGMOD International Conference on Management of Data, Vancouver, Canada, June 10-12, 2008. ACM, 2008. P. 511-524.
70. He B., Yu J.X. High-throughput transaction executions on graphics processors // Proceedings of the VLDB Endowment. 2011. Vol. 4, No. 5. P. 314-325.
71. Heimel M., Saecker M., Pirk H., Manegold S., Markl V. Hardware-oblivious parallelism for in-memory column-stores // Proceedings of the VLDB Endowment. 2013. Vol. 6, No. 9. P. 709-720.
72. Huffman D. A method for the construction of minimum-redundancy codes // Proceedings of the I.R.E. 1952.Vol. 40, No. 9. P. 1098-1101.
73. Idreos S., Groffen F., Nes N., Manegold S., Mullender S., Kersten M. L. MonetDB: Two Decades of Research in Column-oriented Database Architectures // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 40-45.
74. Idreos S., Kersten M. L., Manegold S. Self-organizing tuple reconstruction in column stores // Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, June 29-July 2, 2009, Providence, Rhode Island, USA. ACM, 2009. P. 297-308.
75. Intel Delivers New Architecture for Discovery with Intel Xeon Phi Coprocessors. Intel, 2012. URL: http://files.shareholder.com/downloads/INTC/

0x0x616706/a1067a31-ae32-46a7-a40a-0b842a4c08b9/INTC_News_2012_11_12_Technology_Leadership.pdf (дата обращения: 04.09.2015).

76. Ivanova E., Sokolinsky L. Decomposition of Natural Join Based on Domain-Interval Fragmented Column Indices // Proceedings of the 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO, May 25-29, 2015, Opatija, Croatia. IEEE, 2015. P. 223-226.
77. Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. Morgan Kaufmann Publishers Inc., 2013. 432 p.
78. Jha S., He B., Lu M., Cheng X., Huynh H. P. Improving main memory hash joins on Intel Xeon Phi processors: an experimental approach // Proceedings of the VLDB Endowment. 2015. Vol. 8, No. 6. P. 642-653.
79. Karasalo I., Svensson P. An overview of cantor: a new system for data analysis // Proceedings of the 2nd international workshop on Statistical Database Management (SSDBM'83), September 27-29, 1983, Los Altos, California, USA. Lawrence Berkeley Laboratory, 1983. P. 315-324.
80. Khoshafian S., Copeland G., Jagodis T., Boral H., Valduriez P. A query processing strategy for the decomposed storage model // Proceedings of the Third International Conference on Data Engineering, February 3-5, 1987, Los Angeles, California, USA. IEEE Computer Society, 1987. P. 636-643.
81. Komornicki A., Mullen-Schulz G., Landon D. Roadrunner: Hardware and Software Overview. IBM Red paper. 2009. URL: http://www.redbooks.ibm.com/redpapers/pdfs/redp4477.pdf (дата обращения: 04.09.2015).
82. Lamb A., Fuller M., Varadarajan R., Tran N., Vandier B., Doshi L., Bear C. The Vertica analytic database: C-store 7 years later // Proceedings of the VLDB Endowment. 2012. Vol. 5, No. 12. P. 1790-1801.
83. Larson P.-A., Clinciu C., Fraser C., Hanson E. N., Mokhtar M., Nowakiewicz M., Papadimos V., Price S. L., Rangarajan S., Rusanu R., Saubhasik

M. Enhancements to SQL server column stores // Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13), June 22-27, 2013, New York, NY, USA. ACM, 2013. P. 1159-1168.

84. Larson P.-A., Clinciu C., Hanson E. N., Oks A., Price S. L., Rangarajan S., Surna A., Zhou Q. SQL server column store indexes // Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11), June 12-16, 2011, Athens, Greece. ACM, 2011. P. 1177-1184.
85. Larson P.-A., Hanson E. N., Price S. L. Columnar Storage in SQL Server 2012 // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 15-20.
86. LeHong H., Fenn J. Hype Cycle for Emerging Technologies. Research Report, 2013. URL: http://www.gartner.com/doc/2571624 (дата обращения: 16.12.2014).
87. Lemke C., Sattler K.-U., Faerber F., Zeier A. Speeding up queries in column stores: a case for compression // Proceedings of the 12th international conference on Data warehousing and knowledge discovery, DaWaK'10, August 30-September 2, 2010, Bilbao, Spain. Springer-Verlag Berlin Heidelberg, 2010. P. 117-129.
88. Lima A.A., Furtado C., Valduriez P., Mattoso M. Parallel OLAP Query Processing in Database Clusters with Data Replication // Distributed and Parallel Databases, 2009. Vol. 25, No. 1-2. P. 97-123.
89. Lorie R.A., Symonds A.J. A relational access method for interactive applications // Data Base Systems. Courant Computer Science Symposium. 1971. Vol. 6. Prentice Hall, 1971. P.99-124.
90. MacNicol R., French B. Sybase IQ multiplex – designed for analytics // Proceedings of the Thirtieth International Conference on Very Large Data Bases, August 31-September 3, 2004, Toronto, Canada. Morgan Kaufmann, 2004. P. 1227-1230.

91. Manegold S., Kersten M.L., Boncz P. Database architecture evolution: mammals flourished long before dinosaurs became extinct // Proceedings of the VLDB Endowment, 2009. Vol. 2, No. 2. P. 1648-1653.
92. Mostak T. An overview of MapD (massively parallel database). Massachusetts Institute of Technology white paper, 2013. URL: http://geops.csail.mit.edu/docs/mapdoverview.pdf (дата обращения: 22.04.2013).
93. Naffziger S., Warnock J., Knapp H. SE2 When Processors Hit the Power Wall (or "When the CPU Hits the Fan") // IEEE International Solid-State Circuits Conference, February 10, 2005. ISSCC, 2005. P. 16-17.
94. Neumann T. Efficiently compiling efficient query plans for modern hardware // Proceedings of the VLDB Endowment. 2011. Vol. 4, No. 9. P. 539-550.
95. Nickolls J., Buck I., Garland M., Skadron K. Scalable Parallel Programming with CUDA // Queue. 2008. Vol. 6, No. 2. P. 40-53.
96. NVIDIA's Next Generation Compute Architecture: Fermi. NVIDIA Corporation Whitepaper. 2009. URL: http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf (дата обращения: 04.09.2015).
97. Olsen S., Romoser B., Zong Z. SQLPhi: A SQL-Based Database Engine for Intel Xeon Phi Coprocessors // Proceedings of the 2014 International Conference on Big Data Science and Computing, August 4-7, 2014, Beijing, China. ACM, 2014. Article 17. 6 p.
98. Oueslati W., Akaichi J. A Survey on Data Warehouse Evolution // International Journal of Database Management Systems, 2010. Vol. 2, No. 4. P. 11-24.
99. Pirk H. Efficient cross-device query processing. URL: http://oai.cwi.nl/oai/asset/19964/19964B.pdf (дата обращения 20.10.2015).

100. Plattner H. A common database approach for OLTP and OLAP using an in-memory column database // Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, June 29 – July 2, 2009, Providence, Rhode Island, USA. ACM, 2009. P. 1-2.
101. Plattner H., Zeier A. In-Memory Data Management: An Inflection Point for Enterprise Applications. Springer, 2011. 254 p.
102. Polychroniou O., Raghavan A., Ross K. A. Rethinking SIMD Vectorization for In-Memory Databases // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, May 31-June 4, 2015, Melbourne, Victoria, Australia. ACM, 2015. P. 1493-1508.
103. Pukdesree S., Lacharoj V., Sirisang P. Performance Evaluation of Distributed Database on PC Cluster Computers // WSEAS Transactions on Computers, 2011. Vol. 10, No. 1. P. 21-30.
104. O'Neil P. E., Chen X., O'Neil E. J. Adjoined Dimension Column Index to Improve Star Schema Query Performance // Proceedings of the 24th International Conference on Data Engineering (ICDE 2008), April 7-12, 2008, Cancun, Mexico. IEEE, 2008. P. 1409-1411.
105. O'Neil P. E., O'Neil E. J., Chen X. The Star Schema Benchmark (SSB). URL: http://www.cs.umb.edu/~poneil/StarSchemaB.PDF (дата обращения: 19.10.2015)
106. O'Neil P. E., O'Neil E. J., Chen X., Revilak S. The Star Schema Benchmark and Augmented Fact Table Indexing // Performance Evaluation and Benchmarking, First TPC Technology Conference (TPCTC 2009), August 24-28, 2009, Lyon, France. Springer, 2009. P. 237-252.
107. Padmanabhan S., Malkemus T., Agarwal R., Jhingran A. Block oriented processing of relational database operations in modern computer architectures // Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany. IEEE Computer Society, 2001. P. 567-574.

108. Raducanu B., Boncz P. A., Zukowski M. Micro adaptivity in vectorwise // Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2013), June 22-27, 2013, New York, NY, USA. ACM, 2013. P. 1231-1242.
109. Ramamurthy R., Dewitt D., Su Q. A case for fractured mirrors // Proceedings of the VLDB Endowment. 2002. Vol. 12, No. 2. P. 89-101.
110. Roberts L. G. Beyond Moore's Law: Internet Growth Trends // Computer, 2000. Vol. 33, No. 1. P. 117-119.
111. Roelofs G., Gailly J., Adler M. Zlib: A Massively Spiffy Yet Delicately Unobtrusive Compression Library. URL: http://www.zlib.net/ (дата обращения: 20.09.2015).
112. Roth M. A., Van Horn S. J. Database compression // ACM SIGMOD Record. 1993. Vol. 22, No. 3. P. 31-39.
113. Ruth S.S., Kreutzer P.J. Data Compression for Large Business Files // Datamation. 1972. Vol. 19, No. 9. P. 62-66.
114. Saecker M., Markl V. Big data analytics on modern hardware architectures: a technology survey // Proceedings of the Business Intelligence - Second European Summer School (eBISS 2012), July 15-21, 2012, Brussels, Belgium. Springer, 2013. P. 125-149.
115. SAND CDBMS: A Technological Overview. White Paper. SAND Technology, 2010. 16 p. URL: http://www.sand.com/downloads/side2239eadd/wp_sand_cdbms_technological_overview_en.pdf (дата обращения: 29.10.2015).
116. Schaller R.R. Moore's law: past, present and future // Spectrum, IEEE, 1997. Vol. 34, No. 6. P. 52-59.
117. Scherger M. Design of an In-Memory Database Engine Using Intel Xeon Phi Coprocessors // Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA’14), July 21–24, 2014, Las Vegas, USA. CSREA Press, 2014. P. 21-27.

118. Shapiro M., Miller E. Managing databases with binary large objects // 16th IEEE Symposium on Mass Storage Systems. 1999. P. 185-193.
119. Ślęzak D., Kowalski M. Towards approximate SQL: infobright's approach // Proceedings of the 7th international conference on Rough sets and current trends in computing (RSCTC'10). Springer-Verlag, 2010. P. 630-639.
120. Specification JSON Schema v4. URL: http://json-schema.org/documentation.html (дата обращения: 13.10.2015).
121. Stonebraker M., Abadi D.J., Batkin A., Chen X., Cherniack M., Ferreira M., Lau E., Lin A., Madden S.R., O'Neil E.J., O'Neil P.E., Rasin A., Tran N., Zdonik S.B. C-Store: A Column-Oriented DBMS // Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05), August 30-September 2, 2005, Trondheim, Norway. ACM, 2005. P. 553-564.
122. Stonebraker M., Madden S., Dubey P. Intel "big data" science and technology center vision and execution plan // ACM SIGMOD Record, 2013. Vol. 42, No. 1. P. 44-49.
123. Sun N.-H., Xing J., Huo Z.-G. et al. Dawning Nebulae: A petaflops supercomputer with a heterogeneous structure // Journal of Computer Science and Technology. 2011. Vol. 26, No. 3. P. 352-362.
124. Taniar D., Leung C.H.C., Rahayu W., Goel S. High Performance Parallel Database Processing and Grid Databases. John Wiley & Sons, 2008. 554 p.
125. Thiagarajan S.U., Congdon C., Naik S., Nguyen L.Q. Intel Xeon Phi coprocessor developer's quick start guide. White Paper. Intel, 2013. URL: https://software.intel.com/sites/default/files/managed/ee/4e/intel-xeon-phi-coprocessor-quick-start-developers-guide.pdf (дата обращения: 04.09.2015).
126. Tu Y.-C., Kumar A., Yu D., Rui R., Wheeler R. Data management systems on GPUs: promises and challenges // Proceedings of the 25th International Conference on Scientific and Statistical Database Management

(SSDBM’13), July 29-31, 2013, Baltimore, MD, USA. ACM, 2013. P. 33:1-33:4.

127. Turner V., Gantz J.F., Reinsel D., et al. The Digital Universe of Opportunities: Rich Data and the creasing Value of the Internet of Things. IDC white paper, 2014. URL: http://idcdocserv.com/1678 (дата обращения: 29.01.2015).
128. TOP500: 500 most powerful computer systems in the world. URL: http://www.top500.org (дата обращения: 23.07.2015).
129. TPC Benchmark DS – Standard Specification, Version 1.4.0. Transaction Processing Performance Council (http://www.tpc.org), 2015. 156 p.
130. TPC Benchmark H - Standard Specification, Version 2.17.1. Transaction Processing Performance Council (http://www.tpc.org), 2014. 136 p.
131. Ungerer T., Robič B., Šilc J. A survey of processors with explicit multithreading // ACM Computing Surveys. 2003. Vol. 35, No. 1. P. 29-63.
132. Vanderwiel S.P., Lilja D.J. Data prefetch mechanisms // ACM Computing Surveys (CSUR), 2000. Vol. 32, No. 2. P. 174-199.
133. Viglas S.D. Just-in-time compilation for SQL query processing // Proceedings of the VLDB Endowment. 2013. Vol. 6, No. 11. P. 1190-1191.
134. Wang E., Zhang Q., Shen B., Zhang G., Lu X., Wu Q., Wang Y. High-Performance Computing on the Intel Xeon Phi: How to Fully Exploit MIC Architectures. Springer, 2014. 321 p.
135. Weiss R. A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. Oracle Corporation White Paper, 2012. 35 p. URL: http://www.oracle.com/technetwork/database/exadata/exadata-technical-whitepaper-134575.pdf (дата обращения: 29.10.2015).
136. Westmann T., Kossmann D., Helmer S., Moerkotte G. The implementation and performance of compressed databases // ACM SIGMOD Record, 2000. Vol. 29, No. 3. P. 55-67.

137. Williams M.H., Zhou S. Data Placement in Parallel Database Systems // Parallel database techniques. IEEE Computer society, 1998. P. 203-218.
138. Wynters E. Parallel processing on NVIDIA graphics processing units using CUDA // Journal of Computing Sciences in Colleges. 2011. Vol. 26, No. 3. P. 58-66.
139. Zhang S., He J., He B., Lu M. OmniDB: towards portable and efficient query processing on parallel CPU/GPU architectures // Proceedings of the VLDB Endowment. 2013. Vol. 6, No. 12. P. 1374-1377.
140. Yuan Y., Lee R., Zhang X. The Yin and Yang of processing data warehousing queries on GPU devices // Proceedings of the VLDB Endowment. 2013. Vol. 6, No. 10. P. 817-828.
141. Ziv J., Lempel A., A universal algorithm for sequential data compression // IEEE Transactions on Information Theory. 1977. Vol. 23, No. 3. P. 337-343.
142. Zukowski M., Boncz P. A. Vectorwise: Beyond column stores // IEEE Data Engineering Bulletin. 2012. Vol. 35, No. 1. P. 21-27.
143. Zukowski M., Heman S., Nes N., Boncz P. Super-Scalar RAM-CPU Cache Compression // Proceedings of the 22nd International Conference on Data Engineering, April 3-8, 2006, Atlanta, GA, USA. IEEE Computer Society, 2006. P. 59-71.

ПРИЛОЖЕНИЕ 1. ОСНОВНЫЕ ОБОЗНАЧЕНИЯ

№	Обозначение	Семантика	Страница
1	$R(A, B_1, \ldots, B_u)$	Отношение R с суррогатным ключом A и атрибутами $B_1, \ldots, B_u$, представляющее собой множество кортежей длины $u+1$ вида $(a, b_1, \ldots, b_u)$, где $a \in \mathbb{Z}_{\geq 0}$ и $\forall j \in \{1, \ldots, u\}\left(b_j \in \mathfrak{D}_{B_j}\right)$	37
2	$\mathfrak{D}_B$	Домен атрибута B.	37
3	$\&_R$	Функция разыменования для отношения R.	37
4	$T(R)$	Количество кортежей в отношении R.	38
5	$I_{R.B}$	Колоночный индекс атрибута B отношения R.	38
6	$V_i = \left[v_{i_1}; v_{i_2}\right)$	Фрагментный интервал.	40
7	$\varphi_{\mathfrak{D}_B}$	Доменная функция фрагментации для $\mathfrak{D}_B$.	40
8	$\varphi_{I_{R.B}}$	Доменно-интервальная функция фрагментации для индекса $I_{R.B}$.	41
9	$I_{R.B}^i$	i-ый фрагмент колоночного индекса $I_{R.B}$.	41
10	k	Степень фрагментации.	41
11	$\ddot{\varphi}_{I_{R.C}}$	Функция транзитивной фрагментации индекса $I_{R.C}$ относительно индекса $I_{R.B}$.	43
12	h	Хеш-функция.	66
13	I_h	Колоночный хеш-индекс на базе хеш-функции h.	66
14	φ_{I_h}	Функция фрагментации для колоночного хеш-индекса.	66

ПРИЛОЖЕНИЕ 2. ОПЕРАЦИИ РАСШИРЕННОЙ РЕЛЯЦИОННОЙ АЛГЕБРЫ

№	Обозначение	Название	Описание
1	$R \cup S$	Объединение	Объединение отношений *R* и *S* с одинаковыми схемами.
2	$R \cap S$	Пересечение	Пересечение отношений *R* и *S* с одинаковыми схемами.
3	$\pi_{C_1,\ldots,C_u}(R)$	Проекция	Проекция отношения *R* по атрибутам $C_1,\ldots,C_u$.
4	$\sigma_\theta(R)$	Выборка	Выборка кортежей из отношения *R* по условию θ.
5	$R \times S$	Декартово произведение	Декартово произведение отношений *R* и *S*.
6	$R \bowtie S$	Еестественное соединение	Соединение отношений *R* и *S* по общим атрибутам.
7	$R \underset{\theta}{\bowtie} S$	Тэта-соединение	Соединение отношений *R* и *S* по условию θ.
8	$\gamma_{C_1,C_2,\text{agrf}(D_1,D_2)\to F}(R)$	Группировка	Группировка кортежей отношения $R(C_1,C_2,D_1,D_2)$ по атрибутам C_1, C_2, с последующим вычислением функции агрегирования agrf, применяемой к атрибутам D_1, D_2. Результат агрегирования именуется как атрибут *F*.
9	$\delta(R)$	Удаление дубликатов	Удаление дубликатов кортежей в отношении *R*.
10	$\tau_{C_1,\ldots,C_u}(R)$	Сортировка	Сортировка отношения *R* по атрибутам $C_1,\ldots,C_u$.