Санкт-Петербургский государственный университет
аэрокосмического приборостроения

М.А. Смирнов

Обзор применения методов безущербного сжатия данных в СУБД

Рукопись
Версия от 23.05.2004

Санкт-Петербург
2003-2004

PDF-вариант текста подготовлен специально для
http://compression.ru

По всем вопросам пишите на
ms@compression.ru
Максиму Смирнову.

Содержание

Актуальность темы

Вопрос экономного кодирования информации в системах управления базами данных был поставлен в первой половине 1970-х годов [51, 3], но не потерял актуальности до сих пор. Многие современные исследователи отмечают недостаточную теоретическую проработку проблемы и неэффективность поддержки сжатия данных в промышленных СУБД [14, 24, 59].

Применение в СУБД экономного кодирования без потерь информации приводит к ряду положительных результатов. Наиболее очевидным эффектом является уменьшение физического размера базы данных, журнальных и архивных файлов. Но также часто достигается увеличение скорости выполнения запросов и снижение требований к объему оперативной памяти, что отмечается практически во всех работах в данной области знаний. Поэтому эффективная реализация поддержки сжатия данных существенно улучшает качество СУБД.

Интерес к задаче сжатия данных в СУБД изначально был обусловлен стремлением уменьшить физический объем баз данных. Цена подсистемы ввода-вывода составляла основную часть стоимости аппаратуры. Поэтому при надлежащем интегрировании в СУБД методов безущербного сжатия данных достигалась значительная экономия. Небольшое увеличение числа используемых тактов процессора для кодирования-декодирования более чем компенсировалось снижением затрат на подсистему ввода-вывода. Данный результат применения сжатия востребован и в настоящее время. Действительно, падение относительной стоимости устройств хранения информации на несколько порядков сопровождалось соответствующим увеличением размера типичных баз данных. Тем не менее, в настоящее время фокус интереса все больше смещается на увеличение производительности системы управления данными за счет использования сжатия.

Следует также отметить, что экономное кодирование способствует криптографической защите информации. Устранение статистической избыточности повышает криптостойкость алгоритмов закрытия информации и часто является предварительным действием в схемах шифрования данных.

Использование сжатия данных в СУБД связано с решением множества задач. Любая реализация экономного кодирования в СУБД связана с компромиссом меж-

ду степенью сжатия, требуемым объемом оперативной памяти, числом обращений к внешней памяти и оперативной памяти, вычислительными затратами.

Для применения в СУБД требуются специфические методы и приемы экономного кодирования, поскольку обычные методы не удовлетворяют ряду требований. Практическая ценность реализации достигается только при обеспечении быстрого доступа к произвольной записи или элементу данных. Востребованы так называемые методы сжатия с сохранением упорядоченности, позволяющие выполнять операции сравнения без декодирования данных.

Проблема реализации сжатия в СУБД имеет также такие аспекты, как оптимизация выполнения запросов к сжатым данным, эффективное сжатие результатов выполнения запросов, экономное кодирование метаданных, оценка целесообразности сжатия отдельных элементов, выбор алгоритма сжатия и его параметров с учетом типовых запросов и характера данных.

Данный обзор состоит из двух разделов:

1) рассмотрение состояния исследования проблемы сжатия данных в СУБД, главным образом реляционных;
2) описание поддержки методов сжатия в ряде промышленных реляционных СУБД.

Затронута только проблематика использования методов сжатия без потерь информации. Это объясняется следующим. Во-первых, применение экономного кодирования без потерь информации является более универсальным решением, чем сжатие с потерями. Во-вторых, эта область точнее отвечает интересам автора. В-третьих, приемы сжатия данных с потерями информации (в первую очередь мультимедийных данных) и способы их использования в СУБД обладают существенной спецификой и требуют отдельного рассмотрения. Упомянуты лишь несколько подходов к сжатию данных реляционных таблиц с потерей информации.

Предпринята попытка систематически описать известные подходы к решению проблемы использования безущербного сжатия данных в СУБД, а также выявить недостаточно изученные и перспективные направления работ в проблемной области.

1. Исследования по применению методов сжатия в СУБД

На протяжении более двух десятков лет реляционные СУБД (РСУБД) подтвердили свою жизнеспособность и эффективность. Реляционные системы могут не без успеха заменять специализированные комплексы, например, системы управления многомерными базами данных. Подавляющий сегмент рынка СУБД занят РСУБД. Поэтому основная масса исследований и разработок, затрагивающих вопрос поддержки экономного кодирования в СУБД, явным или неявным образом ориентированы на реляционные системы (РСУБД). Но многие методы сжатия и сопутствующие приемы обработки запросов применимы к базам данных различного типа, например, многомерного и объектного. Исходя из вышеизложенного, данный обзор касается вопросов эффективной поддержки сжатия главным образом в РСУБД. Если излагаемые положения относятся к системам иного типа, это явно указывается.

Следующие несколько параграфов, вплоть до «Содержание и основные тенденции в исследованиях проблемы сжатия данных в СУБД», являются вводными и содержат базовые положения таких областей знаний, как организация РСУБД и методы сжатия данных. Текст основной части обзора опирается на понятия и термины, введенные в этих параграфах.

Общие обозначения

Знаком $\lceil\ \rceil$ показывается округление сверху, знаком $\lfloor\ \rfloor$ — округление снизу.

Запись $< число >_N$ означает, что число записано в системе счисления по основанию N.

Базовые принципы устройства РСУБД. Терминология

Логическая организация

Информация в реляционных базах данных (РБД) логически сохраняется в структурах табличного типа. Отдельная РБД может содержать несколько таблиц. Каждая таблица имеет определенное количество колонок (атрибутов, или столбцов). Отдельная строка таблицы как совокупность значений всех атрибутов однозначно определяется значением одной или нескольких колонок, составляющих так называемый первичный ключ. Например, первичным ключом таблицы, содержащей данные о людях и имеющей столбцы «Номер паспорта», «Полное имя», «Год рождения», может быть колонка «Номер паспорта».

Каждый атрибут имеет определенный тип и соответствующую ему область определения (domain). Атрибут принимает значения из области определения.

Некоторые столбцы могут не содержать определенных значений. Говорят, что такие атрибуты принимают неопределенные значения (отсутствующие, несуществующие). Эти квазизначения обычно обозначаются как NULL.

При описании структуры РБД в данном тексте используются как синонимы следующие термины:

1) таблица, набор данных;
2) строка, запись, кортеж[1];
3) колонка, атрибут, столбец, поле.

Физическая организация

Физическая структура хранения РБД зависит от конкретной СУБД. Базе может соответствовать один или несколько файлов операционной системы. Аналогично, таблице может соответствовать один или несколько файлов. В случае так называемого горизонтального разбиения в разных файлах могут сохраняться строки таблицы, отличающиеся значениями некоторых столбцов. При вертикальном разбиении в отдельные файлы записываются группы колонок таблицы. Но практи-

[1] Кортеж (tuple) — группа взаимосвязанных элементов данных. Термин реляционной алгебры.

чески во всех структурах можно выделить блоки фиксированного размера и регулярного устройства, образующие файлы БД. Вся хранимая информация укладывается в такие блоки. Блок содержит заголовок, собственно данные и является наименьшей порцией данных, пересылаемой при вводе-выводе данных в СУБД на уровне взаимодействия с файловой системой. Такой блок будет далее называться страницей.

Обычно в РБД используется построчный способ сохранения табличной информации. Тогда в одной странице может быть сохранено несколько строк. При этом каждая запоминаемая строка занимает определенное вакантное место (слот). В результате строка таблицы на физическом уровне может быть идентифицирована парой значений <идентификатор страницы, идентификатор слота>.

Для обеспечения эффективности операций ввода-вывода осуществляется буферизация страниц в оперативной памяти. При работе СУБД обычно используется несколько буферов. Совокупность буферов составляет так называемый буферный пул.

Средства обеспечения доступа к данным

В РБД широко используется такое средство логической группировки данных, как реляционное представление данных (view). Представление данных, или просто «представление», — это виртуальная таблица, содержащая только описание данных (метаинформацию). Собственно данные размещаются в обычных таблицах. Представления в большинстве случаев применяются для облегчения разделения прав доступа и упрощения запросов. С функциональной точки зрения свойства представления, как правило, не отличаются от свойств таблицы.

Материализованное представление (materialized view, или snapshot) содержит не только метаинформацию, но и собственно данные в соответствии с состоянием исходных таблиц на момент создания или актуализации представления.

Для ускорения доступа к данным используется индексация данных. Индексные структуры содержат информацию, позволяющую по значению некоторого атрибута или атрибутов получить ссылки на строки таблицы, содержащие такие значения. Наиболее распространенными типами индексов являются B-деревья (B-tree) и битовые карты (bitmap) [2].

Типы РБД

Среди РБД в настоящее время принято выделять два больших класса [2]:

1) базы данных для оперативной обработки транзакций[2] (ООТ) в реальном времени, обеспечивающие сохранение актуальных данных при работе информационных систем;
2) хранилища данных (data warehouse), в которых накапливается в ретроспективе согласованная информация о некоторых процессах с целью дальнейшего ее использования в системах поддержки принятия решений (СППР).

Базы данных для ООТ отличаются высокой степенью нормализации данных, что в значительной степени уменьшает избыточность представления информации и облегчает поддержку согласованности. Как правило, промышленная БД такого ти-

[2] От английского On-Line Transaction Processing, или OLTP.

па содержит сотни и тысячи таблиц малого объема с небольшим количеством атрибутов и имеет сложную структуру.

Хранилища данных, организованные как РБД, обычно имеют простую денормализованную структуру, обеспечивающую высокую скорость выполнения сложных запросов, требующих агрегирования данных и вычисления сводных характеристик. Типовой схемой хранилища данных является так называемая «звезда» (star schema), при которой вся хранимая фактологическая информация записывается в одну большую таблицу фактов (fact table). Каждая строка таблицы фактов соответствует точке в многомерном пространстве, определяемом измерениями хранилища данных. Измерение классифицирует некоторый факт (обычно это число) и, как правило, имеет иерархическую структуру. Типичными измерениями являются время и территория. Для каждого измерения в таблице фактов заводится столбец, в котором в закодированном виде записываются значения измерения для каждой строки. С целью расшифровки закодированных значений для каждого измерения создается отдельная таблица (dimension table). Таблицы связываются на основании закодированных значений измерений так, что таблица фактов становится центром «звезды», а таблицы измерений — ее лучами (см.

Рис. 1). В общем случае хранилище данных имеет несколько таблиц фактов и, следовательно, состоит из нескольких «звезд».

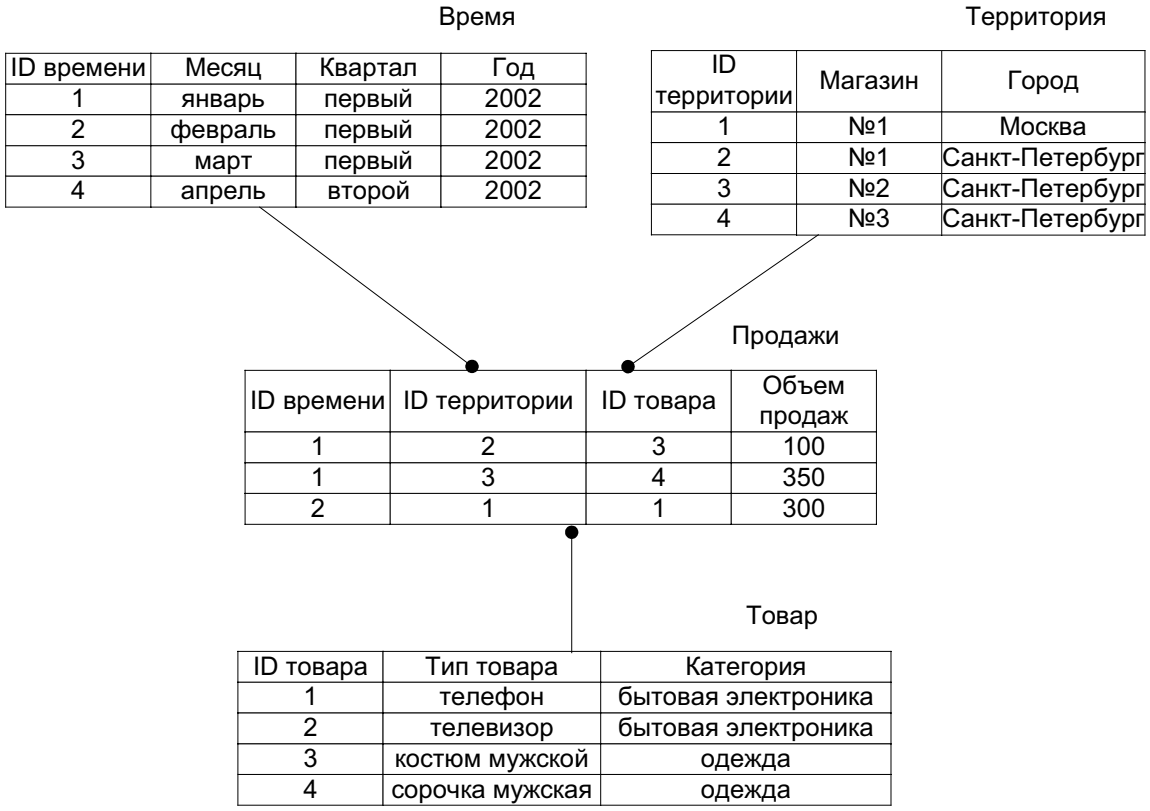

Рис. 1. Пример организации хранилища данных по схеме «звезда»

Стандартные тесты для сравнения СУБД

Наибольшее признание в качестве средств сравнительной оценки производительности транзакционных систем в настоящее время имеют тесты Совета по средствам обработки транзакций (Transaction Processing Council, TPC)[3]. TPC является некоммерческим объединением, в которое входит большинство крупнейших производителей программного и аппаратного обеспечения для серверов СУБД.

Тесты TPC задают набор функциональных требований, которым должна удовлетворять любая транзакционная система вне зависимости от программного и аппаратного обеспечения. Каждый тест включает в себя БД определенного содержания и перечень запросов к ней.

В настоящее время определены следующие тесты TPC.

TPC-A и TPC-B являются устаревшими тестами для систем ООТ.

TPC-C представляет собой действующий тест для систем ООТ. В данном тесте имитируется работа системы учета поступления заказов. Производимые операции включают ввод и пересылку заказов, фиксирование платежей, проверку состояния выполнения заказов, мониторинг запасов товаров на складах. Операции отличаются сложностью и могут как требовать выполнения в реальном времени, так и образовывать очереди для последующего выполнения. Размер БД определяется выбранным коэффициентом масштабирования и равен не менее 80 Мбайт. Производительность измеряется в операциях по формированию заказа, производимых в минуту.

TPC-D является устаревшим тестом для СППР. Минимальный размер тестовой БД составляет порядка 1 Гбайт. TPC-D заменен на TPC-H и TPC/R.

TPC-H предназначен для тестирования СППР. Тест определяет набор специфических запросов к данным по розничной продаже и параллельно производимых модификаций данных. Структура используемых данных соответствует структуре в TPC-D. В зависимости от принятого коэффициента масштабирования размер БД может составлять от 1 до 10000 Гбайт. Производительность определяется по сводной характеристике, измеряемой в запросах в час.

TPC-R сходен с TPC-H, но позволяет производить дополнительную оптимизацию за счет априорного знания запросов.

TPC-W ориентирован на тестирование коммерческих систем с доступом через Web.

Необходимо также отметить тест Wisconsin, разработанный в начале 1980-х и имевший большую популярность в течение долгого времени. В первой половине 1990-х значительно устарел и отдал пальму первенства тестам TPC.

Основные методы сжатия данных. Терминология

Методы сжатия данных без потерь информации основаны на устранении избыточности представления информации. Экономное кодирование достигается за счет представления маловероятных событий более длинными словами, чем событий с высокой вероятностью наступления. Если вероятность наступления события равна p, то, в соответствии с теоремой Шеннона о кодировании источника инфор-

[3] http://www.tpc.org/

мации, такое событие выгоднее всего кодировать словом длиною $-\log_2 p$ битов. Методы сжатия данных явно или неявно опираются на этот факт.

В результате процесса экономного кодирования единице исходных данных (символу, слову, строке, числу и т.п.) ставится в соответствие так называемое кодовое слово. Кодовое слово состоит из последовательности цифр, обычно двоичных. Совокупность всех кодовых слов образует код. Если длины всех кодовых слов одинаковые, то используемый код имеет фиксированную (постоянную) длину, иначе — переменную. Если исходные данные могут быть однозначно восстановлены по массиву соответствующих кодовых слов, то кодирование не приводит к потере информации, т.е. является безущербным, без потерь.

Эффективность сжатия как характеристика сокращения размера представления информации относительно исходного будет в данном обзоре определяться степенью сжатия. Степень сжатия принимается равной отношению объема исходных данных к объему соответствующих им сжатых данных и измеряется в разах.

Все методы сжатия принято разделять на два класса: методы статистического кодирования и методы словарного сжатия. В схемах сжатия также часто используются вспомогательные преобразования, обеспечивающие или способствующие выполнению этапа экономного кодирования.

Статистическое кодирование

Методы статистического кодирования явным образом опираются на теорему Шеннона. Такие методы включают в себя два этапа: оценка вероятности кодируемых элементов (моделирование) и собственно кодирование. На этапе кодирования выполняется замещение элемента s_i с оценкой вероятности появления $q(s_i)$ кодовым словом длиной $-\log_2 q(s_i)$ битов. Этот этап иногда называется энтропийным кодированием. Оценки $q(s_i)$ могут быть получены из безусловных частот встречаемости элементов, условных частот, т.е. с учетом контекста, более сложными способами. Восстановление данных без потерь обеспечивается в том случае, когда кодер и декодер оперируют одними и теми же оценками $q(s_i)$ в каждый момент времени.

Задача кодирования элемента с заданной вероятностью традиционно решается с помощью разновидностей метода Хаффмана и арифметического сжатия [1].

Алгоритм Хаффмана определяет процедуру построения кода переменной длины, средняя избыточность которого минимальна для всех неблочных кодов, т.е. задающих отображение ровно одного исходного элемента в одно кодовое слово. Поскольку слово может быть представлено только целым числом битов, при кодировании по Хаффману осуществляется приближение $q(s_i)$ дробями, равными степеням двойки. Поэтому данный алгоритм неприменим непосредственным образом для экономного кодирования элементов бинарного алфавита. Код Хаффмана является префиксным и поэтому обычно представляется в виде дерева. Традиционно используется двухпроходная схема (статистический алгоритм Хаффмана): при первом просмотре данных подсчитывается статистика встречаемости элементов, т.е. строится модель данных, на основании которой формируется код; при втором просмотре данные сжимаются с помощью полученного кода. Оценки вероятности

$q(s_i)$ при кодировании постоянны, и код не меняется. Известен адаптивный однопроходной вариант алгоритма, но он обладает существенно большей вычислительной сложностью и на практике не используется [23].

Арифметическое сжатие, или арифметическое кодирование, позволяет при кодировании нескольких элементов представлять каждый элемент в среднем дробным числом битов. Поэтому обычно арифметическое сжатие обеспечивает бóльшую степень сжатия, чем кодирование по Хаффману. Блок кодируемых элементов представляется дробью, определяемой произведением оценок вероятности $q(s_i)$ всех элементов блока. Это и определило название метода. Чем $q(s_i)$ меньше, тем длиннее дробь, и больше требуется двоичных символов для ее представления. Алгоритм декодирования сложнее, чем для метода Хаффмана, но позволяет восстановить исходные данные без потерь. Арифметическое кодирование не требует явной перестройки кода при изменении оценок вероятности, поэтому обычно используется адаптивная однопроходная схема, позволяющая естественным образом учитывать локальные особенности данных.

Словарное сжатие

Идея словарного сжатия заключается в замене последовательностей элементов исходных данных на идентификаторы таких фраз некоторого словаря, которые совпадают с замещаемой последовательностью. Методы словарного сжатия эксплуатируют факт повторяемости строк символов. Словарь как совокупность фраз может строиться различным образом. Например, в него могут включаться такие строки, которые обладают наибольшей величиной характеристики $q\left(L-L_I\right)$, где q — частота встречаемости последовательности, L — длина последовательности, L_I — длина идентификатора (указателя) фразы словаря.

Среди словарных схем наибольшее распространении получили методы Зива-Лемпела. Словарные методы, относимые к этому классу, можно разделить на два семейства: LZ77 (LZ1) и LZ78 (LZ2). Схемы семейства LZ77 базируются на одноименном методе, предложенном в [64]. В методах данного семейства роль словаря играет порция уже обработанных данных. Последовательность обычно кодируется указанием позиции начала эквивалентной фразы в словаре (смещения) и длины совпадения. Пара <смещение, длина совпадения> в этом случае является указателем. Если кодируемый элемент отсутствует в словаре, то он некоторым образом помечается и представляется как есть. Такой элемент называется литералом. Из способа формирования словаря следует, что методы семейства LZ77 являются адаптивными.

На практике схемы типа LZ77 используются совместно с алгоритмами статистического кодирования указателей и литералов. Например, в методе LZH для экономного кодирования указателей и литералов используется алгоритм Хаффмана.

В схемах семейства LZ78 в словарь включаются не все последовательности, встреченные в обработанном массиве данных, а лишь «перспективные» с точки зрения вероятности появления в дальнейшем. Например, в методе LZ78 новая фраза формируется как сцепление (конкатенация) одной из фраз S словаря, имеющей самое длинное совпадение с текущей кодируемой последовательностью S', и символа s. Символ s является символом, следующим за последовательностью

$S' = S$ [65]. В отличие от семейства LZ77, в словаре не может быть одинаковых фраз. В самом известном представителе семейства LZ78, методе LZW, словарь инициализируется фразами для всех символов алфавита кодируемых данных[4]. Указатели фраз кодируются словами фиксированной длины, определяемой размером словаря. В рамках метода семейства LZ78 несложно реализовать эффективное неадаптивное и полуадаптивное сжатие, при котором словарь строится заранее.

Словарь, используемый в словарных методах сжатия, можно рассматривать как аналог статистической модели данных, применяемой в статистических методах.

Другие методы сжатия

Распространенным методом сжатия является кодирование длин серий (Run Length Encoding, RLE). Этот метод позволяет кодировать последовательности одинаковых элементов. Существует большое количество разновидностей кодирования длин серий. Например, последовательность идентичных элементов может представляться тройкой <флаг использования кодирования, длина серии, повторяемый элемент> [1].

Для кодирования целых чисел с неизвестным, но монотонно убывающим распределением вероятности используются так называемые универсальные коды. Избыточность универсальных кодов целых чисел для любого конкретного распределения и любого кодируемого числа n является ограниченной сверху, если выполняется условие $p(n) \geq p(n+1)$. Универсальными являются, например, коды Элайеса [1].

Преобразования, используемые в схемах сжатия данных

Сжатие данных может быть улучшено, если обрабатываются не исходные элементы, а их разности. Это характерно, например, для оцифрованных аналоговых сигналов, табличных данных. В последнем случае вычитаются не элементы, а последовательности элементов, т.е. строки таблицы. Для обеспечения возможности однозначного преобразования необходимо сохранить в первоначальном виде первый элемент или любой другой, служащий опорным (базовым). Описанный прием называется дифференциальным или разностным кодированием.

При наличии локальных группировок одинаковых символов сжатие может быть улучшено за счет предварительного преобразования «стопка книг»[5]. Все используемые элементы заносятся в список и кодируются своими номерами в нем. При каждом появлении элемента он переносится в голову списка, «сдвигая» вниз все остальные. В результате локально часто используемые элементы представляются одинаковыми числами, что позволяет применять простые методы сжатия.

---

[4] Welch T.A. A technique for high-performance data compression. *IEEE Computer* 17(6):8-19, Jun. 1984.

[5] Рябко Б.Я. Сжатие данных с помощью стопки книг // Проблемы передачи информации. 1980. Т.16. №4. С. 16-21.

Классификация методов по стратегиям обновления модели (словаря)

Схемы сжатия данных классифицируются также по способу построения и обновления модели (словаря). Выделяют четыре варианта моделирования:

- статическое (неадаптивное);
- полуадаптивное;
- адаптивное (динамическое);
- блочно-адаптивное.

При статическом моделировании для любых обрабатываемых данных используется одна и та же модель (словарь). Иначе говоря, не производится адаптация к особенностям сжимаемых данных.

Полуадаптивное сжатие предполагает, что для кодирования заданной последовательности выбирается *или* строится модель на основании анализа именно обрабатываемых данных.

Адаптивное моделирование является естественной противоположностью статической стратегии. По мере кодирования модель изменяется по заданному алгоритму после сжатия каждого элемента. Однозначность декодирования достигается тем, что, во-первых, изначально кодер и декодер имеют идентичную модель и, во-вторых, модификация модели при сжатии и разжатии осуществляется одинаковым образом.

В случае блочно-адаптивной стратегии обновление модели может выполняться после обработки целого блока элементов, в общем случае переменной длины.

Содержание и основные тенденции в исследованиях проблемы сжатия данных в СУБД

Как было ранее указано, интерес к вопросу сжатия данных в СУБД появился в первой половине 1970-х годов. Первые исследования решали задачи экономного кодирования в файловых системах управления данными. В 1980-х годах, в связи с активным внедрением РСУБД, публикации и разработки уже затрагивали вопросы экономного кодирования информации, хранимой в реляционных базах данных. Исследования 1990-х и последних нескольких лет включают в себя не только вопросы экономного представления собственно хранимых данных, но и сжатия служебной индексной информации, результатов выполнения запросов, эффективной оптимизации запросов с учетом сжатия данных. На Рис. 2 представлено распределение по годам публикаций, использованных при написании этого обзора и имеющих непосредственной отношение к проблеме использования экономного кодирования в СУБД. На Рис. 3 приведено аналогичное распределение, но отдельно показаны публикации, освещающие вопросы сжатия собственно данных и вопросы экономного кодирования индексов[6].

[6] Одна публикация учитывалась два раза, если удовлетворяла двум критериям одновременно.

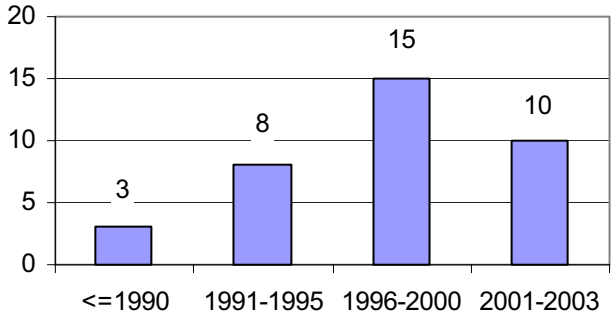

Рис. 2. Распределение публикаций по годам

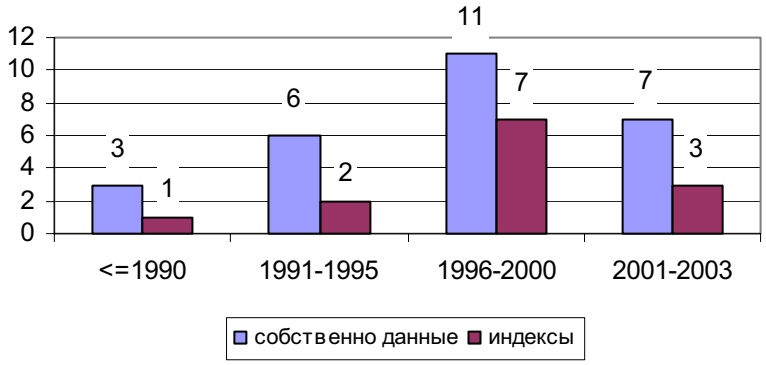

Рис. 3. Распределение публикаций по годам с учетом охватываемой темы: сжатие собственно данных и сжатие индексных структур

Одновременно с изменением направленности и количества исследований смещался и фокус интереса к сжатию. В связи с удешевлением устройств памяти все больший интерес проявляется к такому возможному эффекту, как ускорение обработки запросов при использовании сжатия.

Повышение производительности за счет уменьшения размера хранимых данных обусловлено следующими эффектами [32, 14, 28]:

1) уменьшается среднее время поиска и доступа к информации в устройствах внешней памяти, поскольку для хранения данных требуется меньший объем носителя;
2) повышается фактическая пропускная способность канала ввода-вывода устройства внешней памяти, так как информация о большем количестве записей передается за одну операцию, и снижается количество вспомогательных операций инициации и прекращения обмена;
3) аналогично, повышается фактическая пропускная способность вычислительной сети, что актуально для распределенных баз данных и систем клиент-сервер;
4) уменьшается объем выводимой информации на устройства журналирования, поскольку записи собственно данных и журнальные записи становятся короче;

5) увеличивается фактический размер буфера устройства внешней памяти, и повышается процент успешных попаданий в него;
6) аналогично, увеличивается фактический размер буферного пула СУБД, что позволяет достичь лучшего коэффициента успешных попаданий при неизменном физическом размере буфера или такого же коэффициента при меньшем размере.

Таким образом, повышение быстроты выполнения запросов вследствие использования сжатия вызвано:

- увеличением пропускной способности каналов связи;
- уменьшением времени ожидания извлечения данных из внешней и оперативной памяти за счет того, что больше информации помещается в буфер устройств внешней памяти, оперативную и сверхоперативную (кеш) память.

Использование сжатия данных в СУБД связано с решением множества задач:

- выбор метода сжатия и уровня сжимаемого блока данных в иерархии физической структуры базы данных, т.е. выбор гранулярности блока, сжатие которого осуществляется независимо от других данных; произвольный доступ к данным поддерживается с точностью до такого блока;
- выбор между аппаратным и программным сжатием;
- выбор местоположения модуля сжатия в общей архитектуре вычислительной системы;
- обеспечение обновления данных и выполнения запросов к сжатым данным.

Применение сжатия связано с рядом отрицательных эффектов:

1) увеличение расходов вычислительных ресурсов и, возможно, оперативной памяти;
2) переменная длина записей требует существенного усложнения программного обеспечения;
3) нарушение характеристик данных (например, лексикографической упорядоченности, что требует использования методов сжатия, сохраняющих порядок сортировки);
4) уменьшение возможности успешного восстановления после ошибок.

Для применения в СУБД требуются специфические методы и приемы экономного кодирования, поскольку обычные методы [14]:

1) не обеспечивают требуемой скорости декодирования;
2) работают с блоками достаточно большого объема и не поддерживают произвольный доступ, что не позволяет использовать индексы, в то время как во многих приложениях часто требуется быстрый произвольный доступ (системы ООТ);
3) не учитывают структуру данных; в БД часто сильны вертикальные зависимости между значениями таблицы и слабы горизонтальные; для разных колонок могут быть более подходящими разные методы сжатия;
4) не учитывают статистику по данным, накапливаемую СУБД для последующей оптимизации запросов.

Очень полезным свойством метода сжатия может быть сохранение упорядоченности, что позволяет оперировать при выполнении запросов непосредственно

закодированными данными. Если порядок сортировки закодированных данных соответствует порядку исходных, то можно выполнять операции неэквивалентного сравнения над сжатыми данными (меньше, больше, строгие неравенства). Многие обычные методы сжатия не обладают такой особенностью.

Во многих теоретических исследованиях и практических разработках не уделяется должного внимания целому ряду аспектов проблемы сжатия данных в СУБД. В частности, в [14] указываются следующие вопросы, требующие изучения.

1. Эффективное сжатие текстовых данных. В некоторых работах для сжатия строк используются простые техники типа устранения неопределенных значений или кодирования строковых атрибутов с помощью словаря, содержащего все различные значения атрибута [59]. Но часто неопределенных значений мало, а различающихся значений атрибута много. В других работах используются LZ-методы. В SYBASE IQ используется словарный метод типа LZH для сжатия страницы [25, 57]. В DB2 реализована модификация другого словарного метода, LZW, для сжатия строки [32, 10]. Быстродействие в этих системах страдает от дополнительных издержек на декодирование сжатых строк и атрибутов, не требующихся для выполнения запроса. Техники сжатия, не позволяющие декодировать отдельные строки, требуют сохранения в памяти всей разжатой страницы, что приводит к непрактичному использованию буферного пула. Поэтому необходимо развивать методы сжатия строковых атрибутов.
2. Использование сжатия должны подстраиваться под запросы. Например, не следует сжимать те столбцы, по которым часто выполняется соединение.
3. Результирующее сжатие. Практически не уделяется внимание задаче экономного кодирования результата выполнения запроса. Но это актуально при передаче данных клиенту в случае малой пропускной способности канала связи и строгих ограничениях на объем памяти компьютера-клиента.
4. Оптимизация запросов в СУБД со сжатием. Оптимизация запросов является ключевой проблемой для СУБД со сжатием данным, поскольку эффективность главным образом зависит от того, когда выполняется декодирование при обработке запроса. Мало работ затрагивает эту проблему. Можно указать только [14, 4, 59].

Сжатие табличных данных

Для табличных данных характерны следующие типы избыточности, которые обычно проявляются одновременно [50]:

- разница в частоте встречаемости отдельных символов (элементов); например, если поле текстовое, то число пробелов обычно значительно превышает количество любых других символов;
- наличие последовательностей одинаковых символов; например, это могут быть серии отсутствующих значений или нулей, а также серии пробелов, дополняющие значения строкового типа до получения максимальной ширины, принятой для колонки; последний пример характерен для СУБД, не поддерживающих переменную длину значений строковых атрибутов;
- частое повторение определенных последовательностей символов;

- частое появление символов или последовательностей символов в определенных местах (позициях).

Для табличных данных характерны как вертикальные корреляционные связи (по столбцу), так и горизонтальные (по строке). Часто вертикальные зависимости сильнее горизонтальных. Лучшее сжатие будет обеспечиваться теми методами, которые эффективно эксплуатируют все типы избыточности.

Наиболее распространенными типами данных, используемыми в БД, являются строковые и числовые [14, 24, 32]. Такие типы, как дата и время, могут быть представлены как числовые. Часто методы сжатия данных числовых и строковых (текстовых) типов имеют специфику, поэтому они рассмотрены отдельно.

Кодирование числовых данных

Упаковка битов

Метод упаковки битов (bit packing) заключается в кодировании значения атрибута битовыми последовательностями фиксированной длины. Для каждого сжимаемого столбца требуется хранить соответствующую таблицу перекодировки. Разрядность кодовых последовательностей определяется количеством различных чисел, значения которых реально принимаются атрибутом, или же мощностью области определения атрибута. Очевидно, что требуемая длина кода N равна $\lceil \log_2 n \rceil$, где n — это мощность множества реальных или допустимых значений атрибута. Данный метод рассматривается в одной из самых ранних статей по сжатию баз данных [3]. Метод упоминается наряду с другими приемами в [41] под названием “bit compression”.

Упаковка битов — это простой способ сжатия на уровне значения атрибутов, реализация которого необременительна. Также важно, что при соответствующей реализации сохраняется упорядоченность. Но проблемой при использовании данного метода является обработка появления новых значений атрибута. В этом случае может потребоваться перекодировать все значения столбца, если $\lceil \log_2 n \rceil$ становится больше исходного N. В работах [24, 25] предложена модификация метода для сжатия целых чисел, позволяющая в ряде случаев избежать перекодирования и естественным образом обеспечивающая сохранение упорядоченности. Алгоритм сжатия, названный автором «frame of reference compression» — «сжатие кадра ссылок», состоит из двух частей:

- уровень страницы — сжатие строк и их сохранение в отдельных страницах;
- файловый уровень — перераспределение данных между страницами с учетом способа сжатия уровня страницы для улучшения общей степени компрессии.

Данные, физически перераспределяемые при обработке на файловом уровне, кодируются при обработке на уровне страницы.

Суть метода: для каждого столбца находится минимальное и максимально значение. Значения, попавшие внутрь этого диапазона, последовательно кодируются. Длина кодового слова постоянна и определяется размером диапазона. Пример: пусть имеется последовательность строк из двух полей

S = {(100, 21), (98, 24), (103, 29)}.

Для первого поля диапазон [98, 103], для второго — [21, 29]. Мощность первого множества значений равна 6, второго — 9. Поэтому S будет представлена как:

S = {(010_2, 0000_2), (000_2, 0011_2), (101_2, 1000_2)}.

Очевидно, что если сжатие применяется к полям фиксированной длины, то их новая длина также фиксирована. Для каждого столбца необходимо хранить только границы диапазона и, возможно, длину. Перекодирование требуется лишь в случае выхода значения за границы диапазона.

Степень сжатия сильнее всего зависит от распределения значений каждого атрибута для записей, хранящихся в одной странице. Можно кардинально улучшить сжатие за счет перераспределения записей по страницам при обработке на файловом уровне.

В [24] предлагается применять алгоритм сжатия кадра ссылок и для экономного кодирования индексных структур древовидного вида. В частности, рассмотрено сжатие структур типа R-деревьев[7], в том числе с потерями информации внутри индекса. Последнее позволяет достигать компромисса между используемым объемом внешней памяти для хранения индекса и скоростью поиска. Структура R-дерева такова, что сжатие с потерями не приводит к неоднозначности результатов поиска.

В [25] указывается, что предложенный алгоритм позволил сжать таблицы с данными о фондовых операциях в 3-5 раз. Скорость выполнения запросов при выборке данных в десятки раз превосходила вариант, при котором для сжатия использовался алгоритм типового компрессора Gzip. Экономное кодирование R-деревьев в БД одной геоинформационной системы уменьшило размер их представления до 55% от исходного.

Кодирование целых чисел через разницу между их значением и величиной нижней границы диапазона имеющихся значений атрибута также использовано в [14].

Метод упаковки битов не может обеспечить значительной степени сжатия относительно других методов, что, тем не менее, во многих приложениях может компенсировать высокой скоростью обработки данных и сохранением упорядоченности.

Кодирование длин серий и методы устранение констант

В общем случае, основной выигрыш при кодировании числовых данных может быть получен за счет сжатия ведущих нулей и экономного кодирования часто повторяющихся значений, которыми обычно являются ноль и отсутствующее значение [50, 36, 59]. Такие часто повторяющиеся значения называются в этом подпункте константами. Сжатие ведущих нулей часто выполняется с помощью упаковки битов, рассмотренной в предыдущем подпункте.

Последовательности констант при кодировании длин серий заменяются на тройки <флаг, константа, длина серии> [3, 36]. Существуют другие модификации

[7] R-дерево (R-tree) — сбалансированная древовидная структура данных, разработанная специально для индексирования многомерных пространственных объектов.

метода, но общим недостатком является медленное декодирование, требующее $O(n)$ операций, где n — число элементов исходной строки [36].

Устранение определенной константы может быть реализовано с помощью битовой карты. При этом физически в БД сохраняются только неконстантные значения. Местоположение констант определяется битовой картой, каждый разряд которой равен, например, 1, если в соответствующей позиции находится обычное значение, и 0, если константа. Например, если исходная строка $S = \{D1, D2, c, D3, c, c, c, D4\}$, то сжатая строка $S' = \{D1, D2, D3, D4\}$ и битовая карта $B = \{1,1,0,1,0,0,0,1\}$. Как и для кодирования длин серий, время декодирования равно $O(n)$ [36].

В работах [21, 22] была предложена модификация метода битовой карты, обеспечивающая в среднем более быстрый поиск в сжатых данных. В этом методе, названном «заголовочное сжатие», битовая карта преобразована в заголовочный вектор H, в нечетных позициях которого записывается число несжатых значений с накоплением, а в четных — число пропущенных констант с *накоплением*. Например, для исходной строки $S = \{D1, D2, c, D3, c, c, c, D4\}$ вектор $H = \{2,1,2+1,1+3,2+1+1\} = \{2,1,3,4,4\}$. Декодирование информации может быть выполнено посредством бинарного поиска по H, поэтому время декодирования пропорционально $O(\log|H|)$ [36].

В [36] описана более сложная модификация метода битовой карты, позволяющая сократить число обращений к внешней памяти при декодировании. Метод назван “BAP”, поскольку при сжатии используется три объекта: битовый вектор (bit vector, BV), адресный вектор (address vector, AV) и так называемый физический вектор (physical vector, PV). В позиции битового вектора записывается 0, если текущий символ равен константе, и 1, если наоборот. В физическом векторе перечисляются все неконстантные значения. Битовый вектор разбивается на подвектора, каждый из которых независимо сжимается с помощью кода Голомба[8] и сохраняется в отдельном блоке или последовательности блоков устройства внешней памяти. В ячейке $AV(i)$ адресного вектора хранится относительная позиция в физическом векторе последнего неконстантного элемента для подвектора с номером $i-1$. Первый элемент адресного вектора равен нулю. Если в предыдущем подвекторе одни константы, то в адресный вектор записывается значение его предыдущего элемента. Адресный вектор также может быть сжат с помощью дифференциального кодирования. При кодировании и декодировании на основании адресного вектора определяется, какой подвектор надо распаковать, и декодируется только он. Если адресный вектор достаточно мал, чтобы поместиться в оперативную память, то при декодировании требуется только одно обращение к внешней памяти.

Например, если размер подвектора равен 5, и константа есть “0”, то для строки

---

[8] Код Голомба позволяет оптимальным образом закодировать неотрицательные целые числа, имеющие геометрическое распределение. В данном случае числа — это длины серий нулей. Код предложен в [S.W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, vol. 12, pp. 399-401, July 1966]. Доказательство оптимальности имеется в [R. Gallager and D.V. Voorhis. Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory*, vol. 21, pp. 228-230, Mar. 1975].

$S = \{1,0,0,4,0,\ 0,0,0,0,0,\ 0,0,8,0,0,\ 0,12,0,0,0,\ 0,17,0,0,20\}$

содержимое векторов будет таким:

$BV = \{1,0,0,1,0,\ 0,0,0,0,0,\ 0,0,1,0,0,\ 0,1,0,0,0,\ 0,1,0,0,1\}$,

$PV = \{1,4,8,12,17,20\}$,

$AV = \{0,2,2,3,4\}$.

В частности, $AV(5) = 4$, поскольку последний неконстантный элемент 4-го подвектора содержится в $PV(4)$. Это число 12.

Варьируя параметры, можно задавать соотношения между числом обращений, размером адресного вектора, размером блока, максимальной степенью сжатия и максимальным размером БД.

В [36] приведены результаты практического сравнения методов по степени сжатия. BAP выиграл у битовой карты в рассмотренных случаях. BAP работал стабильнее кодирования длин серий и сжатия заголовков, выигрывая у этих методов при отсутствии кластеризации неконстантных значений или констант в непрерывные серии по несколько сотен символов и более.

Следует подчеркнуть, что экономное кодирование констант актуально только для сильно разреженных БД. Например, такое свойство характерно для БД, в которых накапливаются данные о физических экспериментах.

Статистическое кодирование

Статистическое кодирование именно числовых данных редко упоминается в публикациях. Методы арифметического сжатия, кодирования по Хаффману и другие способы статистического кодирования применялись к данным произвольного типа. В [3] рассмотрена простая схема сжатия чисел посредством использования кода переменной длины, строящегося на основании статистики встречаемости чисел. Множество чисел разбивается на группы в соответствии с частотой использования так, что в группы малого размера попадают часто используемые значения. Элементы в пределах группы кодируются словами одинаковой длины $\lceil \log_2 n \rceil$, где n — размер группы. Кодовые слова разных групп различаются за счет использования флагов или префиксов. Такая схема обеспечивает в общем случае худшее сжатие, чем алгоритм Хаффмана, но может требовать меньше времени для кодирования.

Статистическое кодирование в силу достаточного затратного декодирования может иметь преимущество только при большой статистической избыточности числового массива и необходимости обеспечения высокой степени сжатия. Последнее требует применения сравнительно сложного моделирования, иначе более выгодным будет словарный подход к сжатию.

Дифференциальное кодирование

При обычном дифференциальном кодировании на уровне строки значение $T[i, j]$ i-ой строки j-го столбца заменяется на разность $T[i-1, j] - T[i, j]$. Если в пределах колонки имеется значимая тенденция в изменении данных, то результирующие разности могут быть экономно представлены с помощью простых методов.

В [37] описано применение дифференциального кодирования для сжатия БД с информацией о веществах-кандидатах для создания лекарств. БД используется для обеспечения процесса поиска лекарства для заданной болезни. Каждое вещество может характеризоваться тысячами признаков. Необходимость хранить описания нескольких миллионов веществ приводит к получению БД объемом в десятки гигабайт, что обуславливает актуальность экономного кодирования для уменьшения размера БД и повышения скорости выполнения запросов. Основная таблица БД состоит из трех числовых столбцов: номер вещества-кандидата, номер признака, собственно значение признака (может принимать отсутствующее значение). Значение признака вещества вещественное. При работе с БД используется как, главным образом, случайный доступ, так и последовательный. Новые записи вносятся редко и сразу большими массивами.

Отмечается, что использование схемы дифференциального кодирования, предложенной в [41], непрактично для рассматриваемой БД, поскольку в указанном алгоритме предполагается, что области определения атрибутов небольшие по мощности и заданы заранее. Предложенная схема сжатия, названная «сжатие окна» (window-based compression), включает в себя следующие шаги при кодировании.

1) Сортировка данных по номеру вещества и номеру признака.
2) Группировка соседних записей в так называемые окна. За счет сортировки значения столбцов «номер вещества» и «номер признака» всегда располагаются в возрастающем порядке внутри окна, и разница между ними мала.
3) Значения столбцов «номер вещества» и «номер признака» заменяются на их разность с соответствующими значениями первой строки окна, которая остается в исходном виде.
4) Полученные строки кодируются. Каждый символ представляется 4-битовым кодовым словом. При этом наиболее часто встречаемая пара символов кодируется одним словом.

Размер окна было решено выбрать переменным, так что в пределах одного окна все записи имеют одинаковое значение столбца с номером вещества.

Оценка эффективности сжатия проводилась с помощью специально разработанной СУБД, поддерживающей обновление записей и индексирование данных. Было произведено сравнение эффективности применения кода постоянной длины и кода Хаффмана для шага 4. Указывается, что использование кода Хаффмана не привело к повышению общей производительности работы с БД. Степень сжатия типовой БД веществ-кандидатов при использовании предложенной схемы составила 3.55 раза, при этом, для сравнения, степень сжатия для компрессора Gzip равнялась 5.3 раза. Приведены следующие характеристики изменения скорости выполнения запросов, считающих данные, из-за использования сжатия:

- при отсутствии индексов скорость для случайного доступа в 4 раза выше относительно скорости для несжатой БД, скорость для последовательного — в 2 раза;
- при наличии индексов скорость случайного доступа на 300-20% выше скорости при несжатой БД (чем больше число читаемых записей, тем разница меньше), скорость последовательного — ниже на 10-15%.

Для сравнения указывается, что запросы, требующие случайного доступа, выполняются посредством разработанной тестовой СУБД в 2 раза быстрее, чем Oracle 8*i*. Известно, что в данной версии Oracle поддерживается только сжатие индексов (см. параграф «Oracle» раздела «2. Использование сжатия данных в современных СУБД»).

Делается вывод, что сжатие использовать не нужно, если чтение из индексированной базы главным образом последовательное. Если используется случайный доступ, то сжатие может троекратно увеличить скорость.

Дифференциальное кодирование как средство учета вертикальных зависимостей является одним из основных способов сжатия данных в СУБД и часто используется в качестве одного из шагов алгоритма сжатия. Главный недостаток метода состоит в сложности обеспечения случайного доступа, поскольку для декодирования необходимо знать базовое значение.

Кодирование текстовых данных и данных произвольного типа

Метод Хаффмана

Метод Хаффмана как один из основных и наиболее популярных способов экономного кодирования неоднократно рассматривался применительно к задаче сжатия в СУБД.

В [51] описывается использование кодирования по Хаффману для решения практической задачи сжатия БД с заявками на приобретение продукции. Требовалось обеспечить степень сжатия в 2 и более раз, программа реализации кодирования и декодирования должна была иметь малый размер, устанавливалось жесткое ограничение на время декодирования записи. БД содержала колонки как строкового, так и числового типов, причем значения столбцов редко повторялись. В качестве кандидатов рассматривались варианты кодирования длин серий, словарного кодирования, схемы статистического сжатия. В итоге всем ограничениям удовлетворил только неадаптивный алгоритм Хаффмана с расширением алфавита символов. Для улучшения сжатия к 64 символам алфавита были добавлены 12 часто встречающихся последовательностей символов. В результате была достигнута степень сжатия 2.58 раза. При этом ключевые столбцы и те столбцы, по значениям которых часто производились выборки, были оставлены незакодированными. Словарный алгоритм, продемонстрировавший аналогичную степень сжатия, был отсеян по критериям скорости декодирования и размеру программы.

В [3] метод Хаффмана рассматривается наряду с другими более простыми приемами сжатия табличных данных.

В [20] для улучшения сжатия табличных данных по методу Хаффмана было предложено использовать отдельные распределения частот для различных типов данных, получаемые на основании выборок из таблиц. Для обеспечения большей скорости разжатия использована модификация кода Хаффмана с длиной слов, кратной байту (максимум 2 байта). Сжатие производилось на уровне строки, т.е. не обеспечивался доступ к закодированным значениям отдельных атрибутов. Выбор распределения частот и, соответственно, кода Хаффмана для сжатия очередного элемента производился на основании предыдущего обработанного символа. Если предыдущий символ причислялся к типу данных T, то для кодирования текущего

символа использовалось распределение частот для T. Указано, что степень сжатия для предложенного алгоритма составила 2 раза, количество операций ввода-вывода уменьшилось на 32.7%, при этом число используемых тактов процессоров выросло только на 17%. Данный алгоритм использовался в СУБД IMS фирмы IBM [20, 32].

В чистом виде метод Хаффмана не может конкурировать со словарными схемами по степени сжатия, что предопределяет его нишу вспомогательной техники. Достоинство подхода в простоте и скорости декодирования. Метод допускает простую аппаратную реализацию.

Арифметическое сжатие

В работе [48] на основе сравнительного анализа эффективности различных схем сжатия было установлено, что полуадаптивное арифметическое кодирование обладает преимуществом в изученных ситуациях с точки зрения обеспечиваемого сжатия. Автором работы предлагается использовать в СУБД алгоритм арифметического кодирования на уровне значения атрибута с отдельной моделью для каждого столбца.

Сравнивались методы: кодирование длин серий (КДС), адаптивный вариант LZW, адаптивное и полуадаптивное арифметическое кодирование (ААК и ПАК), адаптивное и полуадаптивное кодирование по Хаффману (АКХ и ПКХ). Сопоставление производилось на трех наборах данных, описанных в Таблица 1, и при различных уровнях гранулярности. Размер страницы был положен равным 4 кБайт. Полуадаптивные алгоритмы ПАК и ПКХ опирались на статистику, собранную со всей таблицы. Для ПАК также рассмотрен вариант, при котором статистика собиралась отдельно для каждого столбца. В этом случае использовалось несколько распределений. Результаты сравнения по степени сжатия представлены в Таблица 2. Приводимые сведения интересны в первую очередь тем, что позволяют оценить влияние гранулярности на степень сжатия данных разных типов.

Таблица 1

Название	Описание таблицы
PUB	Библиография научных публикаций. 11 строковых атрибутов (имена авторов, наименование, издательство и т.п.) и 80690 записей. Длина записи 1828 байтов, при этом средняя длина значений большинства строковых атрибутов значительно меньше максимальной ширины соответствующих столбцов.
BANK	Банковская информация о счетах клиентов. 24 атрибута, из которых 19 строковых и 5 числовых, и 55862 записи. Длина записи 236 байтов.
SYN	Искусственно сгенерированная таблица. 12 атрибутов, из которых 11 числовых и 1 строковый, и 9964 записи. Длина записи 76 байтов. При генерировании числовых значений использовались распределения, полученные из статистики по некоторым таблицам с реальными данными.

Таблица 2

Таблица	Метод	Степень сжатия при разных уровнях гранулярности				
		Файл	Страница	Строка	Значение атрибута	Значение атрибута, разные распределения
PUB	КДС	5.09	5.00	**4.78**	**4.44**	
	LZW	**11.86**	**5.85**	**4.80**	3.36	
	ААК	4.87	4.18	3.62	2.50	
	ПАК	4.72	4.69	4.55	4.27	**4.89**
	АКХ	4.10	3.84	3.55	2.99	
	ПКХ	3.99	3.98	3.91	3.60	
BANK	КДС	1.32	1.32	1.28	1.12	
	LZW	**5.26**	**2.86**	1.58	0.84	
	ААК	2.06	1.94	1.42	0.94	
	ПАК	1.87	1.86	**1.81**	1.36	**1.79**
	АКХ	1.99	1.90	1.55	0.97	
	ПКХ	1.77	1.77	1.70	**1.40**	
SYN	КДС	1.27	1.27	1.23	1.04	
	LZW	**1.86**	1.68	1.26	0.75	
	ААК	1.79	1.74	1.26	0.83	
	ПАК	1.80	**1.80**	**1.67**	**1.24**	**1.27**
	АКХ	1.79	1.63	1.16	0.97	
	ПКХ	1.78	**1.78**	**1.66**	**1.26**	

Из Таблица 2 следует, что в случае текстовых полей сжатие достигается в первую очередь за счет экономного кодирования оконечных пробелов, принудительно увеличивающих длину значения до заданной ширины столбца. Если СУБД поддерживает строковые атрибуты переменной длиной, то эффект от КДС должен становиться в большинстве случаев крайне незначительным. LZW плохо работает на таблицах с колонками малой ширины. Адаптивные методы ААК и АКХ неэффективны на уровне страницы и приводят к увеличению объема данных при сжатии на уровне значений атрибутов. При малых уровнях гранулярности — строка и столбец, наилучшим образом подходящих для эффективного выполнения запросов, стабильно хорошие результаты показывают ПАК и ПКХ. Поскольку ПАК обеспечивает существенно лучшее сжатие текстовых данных, автор [48] выбирает этот метод в качестве базового для создания новой схемы сжатия. В этой схеме, названной в [49] “COLA” (Column-Based Attribute Level Non-Adaptive Arithmetic Coding — «неадаптивное арифметическое кодирование уровня атрибута по колонкам»), значение атрибута арифметически кодируется на основании распределения частот, полученного для данного атрибута. Характеристики степени сжатия для такой схемы представлены в последней колонке Таблица 2.

Тем не менее, в [48] не дается сравнительных оценок скорости кодирования и, главное, декодирования. Поскольку декодирование для арифметического сжатия сложнее, чем для алгоритма Хаффмана, то выбор в пользу арифметического кодирования представляется недостаточно обоснованным. Характеристики производительности являются первостепенными. Например, в [32] при выборе алгоритма сжатия для СУБД DB2 предпочтение было отдано методу семейства Зива-Лемпела,

который при сходной степени сжатия обеспечивал бóльшую скорость декодирования, чем использованная схема арифметического сжатия.

В [8] имеется краткое упоминание об эксперименте по применению арифметического кодирования для сжатия БД. Для учета неоднородности данных в пределах строки, обусловленной разным характером данных отдельных столбцов, авторы предложили использовать несколько алфавитов. Это позволяет ускорить процесс адаптации к локальным особенностям информации. Отмечено увеличение степени сжатия на 12% при тестировании на файлах СУБД Ingres.

Следует отметить, что использование арифметического кодирования в СУБД может быть перспективным, поскольку арифметическое кодирование сохраняет порядок сортировки элементов в соответствии с их накопленной вероятностью, если упорядоченность не жертвуется в пользу ускорения обработки [6, 63]. В [35] описана модификация алгоритма арифметического кодирования для решения задачи сжатия исполнимого кода с возможностью декодирования «на лету». Разработанный алгоритм обеспечивает случайный доступ к сжатым данным с точностью до блока задаваемого размера. Предложенная схема позволила получить бóльшую степень сжатия, чем алгоритм Хаффмана, при сходной скорости декодирования. При этом использовалась полуадаптивная модель, размер блока составлял 4..64 байта. Очевидно, что разработанный алгоритм соответствует требованиям, предъявляемым к методам экономного кодирования для их использования в СУБД.

Методы Зива-Лемпела

Вопрос использования методов Зива-Лемпела для сжатия данных в СУБД исследован, в частности, в работах [32, 48]. Авторы приходят к заключению, что методы Зива-Лемпела непрактичны для сжатия на уровне значения столбца. В [32] также указывается, что адаптивные методы проигрывают по степени сжатия неадаптивным (полуадаптивным).

В [32] для использования в СУБД предложен вариант метода Миллера-Уэгнама (Miller-Wegnam), или LZMW [1]. В методе Миллера-Уэгнама каждая фраза словаря представляет собой конкатенацию двух других фраз, а не фразы и символа алфавита данных, что отличает метод от LZW. Сжатие в предложенной схеме выполняется на уровне строки и является полуадаптивным. Словарь строится при загрузке данных и может иметь размер до 4096 фраз. Указывается, что использование сжатия позволило сократить размер типовых БД в 2 раза при аналогичном увеличении скорости выполнения запросов. Расход тактов процессора увеличился всего на 20%. Данная схема была реализована в СУБД DB2 фирмы IBM. Подробнее см. параграф «DB2» раздела «2. Использование сжатия данных в современных СУБД».

Методы Зива-Лемпела часто использовались как эталонные при сравнении схем сжатия данных. Так, в [41] производится сравнение эффективности LZW с другими методами сжатия применительно к задаче экономного кодирования в СУБД. Аналогичную роль играет компрессор Gzip, реализующий метод LZH, в [24, 11, 37].

В [63] рассмотрена схема модификации метода LZW, позволяющая достичь свойства сохранения упорядоченности. Упорядоченность теряется при назначении разных кодовых слов строкам, которые являются префиксами друг друга. Поэтому

для обеспечения сохранения порядка сортировки в словарь добавляются фразы, образуемые слиянием имеющихся фраз и «пустого» символа (“zilch”), не принадлежащего алфавиту данных. Эта модификация выполняется только для фраз, которые являются префиксами других фраз словаря. Указано, что предложенный прием применим к любым словарным схемам типа LZ78, задающим отображение исходной последовательности переменной длины в кодовое слово фиксированный длины, если все символы алфавита данных являются фразами словаря.

Основным недостатком методов Зива-Лемпела является сложность обеспечения произвольного доступа к информации.

Другие способы словарного сжатия

Сжатие табличных данных посредством словарного кодирования на уровне значений атрибутов часто изучалось в исследованиях [3, 14, 17, 59]. Существует ряд практических реализаций таких схем сжатия в СУБД [40, 46]. В большинстве случаев словарь для сжатия отдельного столбца образуется из значений данного атрибута. Обычно фразами словаря делаются все имеющиеся значения. Кодирование при этом сводится к замене значения на номер соответствующей фразы словаря (указатель).

В диссертационной работе [14] предложен подход к сжатию строковых атрибутов под названием «иерархическое словарное кодирование», в котором используется набор из конкурирующих алгоритмов сжатия, работающих на разном уровне гранулярности данных. Используются полуадаптивные алгоритмы словарного кодирования на уровне целого значения атрибута, которое часто является последовательностью слов, слова, префикса/суффикса слова и символа. Уровень детальности выбирается исходя из минимизации суммы размеров закодированной колонки и словаря. Таким образом, обеспечивается возможность эффективного учета различных типов избыточности. Для оценки жизнеспособности подхода было произведено его сравнение с другими словарными методами на данных теста TPC-H. В сопоставлении были задействованы реализации полуадаптивного алгоритма LZW, кодирования по словарю значений атрибута и кодирования по словарю из слов, встречающихся в значениях атрибута. Предложенный подход имел наибольшую эффективность среди рассмотренных методов, обеспечив двукратное сжатие при увеличении скорости последовательного доступа к данным на 30%.

Дифференциальное кодирование, векторное квантование

Дифференциальное кодирование для преобразования данных произвольного типа упоминается в обзорной статье [50]. Идея использования дифференциального кодирования получила развитие в рамках подхода по применению векторного квантования для преобразования и сжатия табличных данных [41, 43, 44].

Векторное квантование можно рассматривать как расширение дифференциального кодирования для многомерного случая. Суть векторного квантования заключается в разбиении множества объекта на группы по некоторому критерию сходства. Среди объектов каждой группы выбирается один, выражающий наиболее характерные свойства объектов группы (образцовый объект). В результате векторного квантования все объекты группы приравниваются к образцовому. Если требуется выполнить преобразование без потерь, то сохраняется разница между текущим

и образцовым объектами. Это и определяет родство с дифференциальным кодированием.

В [43, 44] предложен способ использования векторного квантования для безущербного сжатия в СУБД. Способ был разработан для экономного кодирования баз статистических данных. В [41] подход к сжатию был пересмотрен, и результирующий алгоритм стал фактически эквивалентеным дифференциальному кодированию.

Отдельная строка таблицы рассматривается как точка в n-мерном пространстве R, образованном декартовым произведением всех n атрибутов. Например, если имеются два атрибута, и их множества значений $A_1 = \{a, b\}$ и $A_2 = \{0, 1, 2\}$, то пространство состоит из 6 точек: $R = \{(a,0), (a,1), (a,2), (b,0), (b,1), (b,2)\}$. Очевидно, что мощность $|R| = \prod_{i=1}^{n} |A_i|$.

Процедура сжатия состоит из нескольких шагов.

1. Все строки упорядочиваются в соответствии с правилами упорядочивания отдельных атрибутов. Например, если атрибут строковый, то его значения располагаются в словарном (лексикографическом) порядке. Каждой строке ставится в соответствие ее порядковый номер в полученной последовательности кортежей. Иначе говоря, с помощью некоторой функции $\varphi : R \to N_R$, где $N_R = \{1, 2, \ldots |R| - 1\}$, производится отображение $R \to N_R$, каждой строке $t \in R$ ставится в соответствие $\varphi(t)$. Для обеспечения лучшей результирующей степени сжатия рекомендуется составлять φ так, чтобы сначала сортировать строки по атрибутам с большим количеством значений.
2. Массив строк разбивается на блоки. Размер блока принимается равным величине страницы таблицы. Внутри блока все строки упорядочены.
3. Блоки кодируются. Первая строка t_1 оставляется в исходном виде, т.е. выполняет роль образца. Другие строки $t_i, i > 1,$ кодируются через разность $\varphi(t_i) - \varphi(t_{i-1}) > 0$. Разность записывается в позиционной системе счисления с весами $\left\{ \prod_{i=1}^{n-1} |A_i|, \prod_{i=1}^{n-2} |A_i|, \ldots, |A_n|, 1 \right\}$. Для улучшения сжатия последовательность ведущих нулей кодируется через указание ее длины. Например, если $\varphi(t_i) - \varphi(t_1) = 567$, $n = 4$, мощности множеств $|A_k|$ равны 4, 3, 64, 100 соответственно, то разность представляется как $\{2, 5, 67\}$, поскольку $567 = 0 \cdot 3 \cdot 64 \cdot 100 + 0 \cdot 64 \cdot 100 + 5 \cdot 100 + 67 \cdot 1$. Два ведущих нуля обозначены первой цифрой.

Если атрибуты не являются целыми числами, то для преобразований $R \to N_R$ и $N_R \to R$ может потребоваться поддерживать таблицу соответствия между значением атрибута и его номером в отсортированной последовательности значений. В [41] предлагается для каждого такого атрибута создавать словарь из всех используемых значений.

Таким образом, сжатие достигается за счет:

1) упаковки битов (или словарного перекодирования) при переходе от значений атрибутов к порядковым номерам;
2) кодирования длины серии ведущих нулей, часто образующейся в результате взятия разности.

В [41] приведены оценки эффективности предложенного подхода для БД результатов обследования домохозяйств Бюро переписи населения США. Указывается степень сжатия 4 раза при увеличении скорости выполнения запросов на 41.3%. Использование для сжатия данных алгоритма LZW привело к получению аналогичной степени сжатия, но повышение скорости работы составило только 35.2%.

В более ранней работе [43] в качестве образцовой строки выбиралась строка t_k, находящаяся посередине блока, а прочие строки t_i кодировались через разность $\begin{cases}\varphi(t_k)-\varphi(t_i), i<k\\ \varphi(t_i)-\varphi(t_k), i>k\end{cases}$. Это точнее соответствует сути векторного квантования. Для повышения степени сжатия выполнялся второй этап взятия разности: текущая строка характеризовалась результатом вычитания ее разности из разности предыдущей строки. Из сравнения результатов экспериментов, приводимых в [43] и [41], следует, что при использовании чистого векторного квантования степень сжатия меньше примерно на 10%, а скорость декодирования ниже на несколько десятков процентов. Таким образом, применение дифференциального кодирования оказалось более предпочтительным в рамках проведенных исследований.

Как уже отмечалось, векторное квантование и дифференциальное кодирование актуальны как средства простого учета вертикальных взаимосвязей в данных, но при их применении усложняется организация эффективного произвольного доступа к информации.

Сортировка, группировка и преобразование столбцов

Сортировка значений как предварительная операция позволяет применять к полученным данным простые методы сжатия. Но при сжатии без потерь необходимо иметь возможность восстановить исходный порядок при декодировании.

Сортировка в пределах колонки или набора колонок C может быть легко обращена, если эти колонки функционально зависят от известной опорной группы колонок P. Тогда значения C сортируются по значениям P. В этом случае лексикографически отсортированные значения P дадут индекс, позволяющий обратить сортировку. В примере ниже $P=\{\text{ID}\}$, $C=\{\text{Город}\}$.

№ строки	ID	Город	Отсортированная опорная колонка "ID"	Колонка "Город", отсортированная по ID	Индекс для восстановления
1	1	Москва	1	Москва	1
2	2	Санкт-Петербург	1	Москва	3
3	1	Москва	1	Москва	4
4	1	Москва	1	Москва	6
5	2	Санкт-Петербург	2	Санкт-Петербург	2
6	1	Москва	2	Санкт-Петербург	5

Для однозначного восстановления сортировка должна быть устойчивой, с сохранением порядка одинаковых элементов. Стоит отметить, что такой прием используется и при обратном преобразовании Барроуза-Уилера [1].

В [58] предложен эффективный субоптимальный алгоритм нахождения дерева зависимости групп колонок от некоторой пары других колонок. Оценка полезности сортировки основана на подсчете длин серий, возникающих при сортировке. Авторы указывают, что учет функциональной зависимости только максимум от $k = 2$ колонок хорошо работает на практике. В то же время задача определения оптимального разбиения при $k \geq 2$ является НП-трудной, т.е. число шагов алгоритма полиноминально растет с увеличением числа столбцов. Показано, что предложенная схема позволяет кардинально улучшить сжатие табличных файлов не только относительно универсальных компрессоров, но и специализированного табличного компрессора, описанного в [11].

Известно, что перестановка столбцов может существенно улучшить сжатие при построчной обработке. Для нахождения оптимальных группировок в [11] предложено кодировать небольшой обучающий фрагмент данных и использовать статистику для решения задачи с помощью динамического планирования (программирования). Для ускорения процедуры оптимизации n столбцов разбиваются на k попарно непересекающихся групп. Это позволяет получить субоптимальный результат при временной сложности $O(nk)$ вместо $O(n^2)$. Для исследованных данных потери в результирующей степени сжатия крайне малы в том случае, когда n и k отличаются только в разы, т.е. $n/k < 10$.

В [11] также рассмотрено дифференциальное кодирование на уровне столбцов при сжатии данных произвольного типа. Для данных, рассматриваемых в указанной публикации, характерно, что значения $T[j]$ некоторого исходного столбца j таблицы T сжимаются слабее, чем результат преобразования $T[i]-T[j]$, где i – опорный столбец. Взятие $T[i]-T[j]$ используется как предварительный этап перед кодированием с помощью LZH.

Другие методы сжатия данных

Способ экономного кодирования данных на основе грамматической модели был предложен в [12]. Созданный алгоритм, названный авторами RAY, основан на достаточно известном алгоритме универсального сжатия данных SEQUITUR[9]. В SEQUITUR реализуется автоматическое адаптивное построение контекстно-свободной грамматики для обрабатываемых данных. По сути SEQUITUR является словарной схемой сжатия, в которой каждая фраза словаря соответствует некоторому правилу грамматики.

В отличие от SEQUITUR, RAY является многопроходным полуадаптивным алгоритмом. В результате каждого этапа просмотра данных выполняется модификация грамматической модели: добавляются новые правила и изменяются при необходимости существующие. Утверждается, что многопроходное построение модели позволяет сократить расходы оперативной памяти и обеспечивает возможность выбора между скоростью и степенью сжатия за счет варьирования числа этапов. Как и в SEQUITUR, минимальная длина правой части грамматического правила (продукции) равна двум символам. Но, если в SEQUITUR для создания правила требуется двукратное появление пары, в RAY используется переменный порог час-

[9] См., например: Nevill-Manning C. Inferring Sequential Structure. *PhD Thesis*, University of Waikato, May 1996.

тоты встречаемости во всем массиве данных. Величина порога зависит от номера этапа. После каждого этапа правила грамматики кодируются по алгоритму Хаффмана. Собственно обрабатываемая последовательность, которая является первым правилом грамматики и включает в себя ссылки на другие правила (т.е. номера правил), кодируется сходным образом.

В [12] приведены результаты сравнения RAY с другими алгоритмами на данных трех типов. Отмеченная степень сжатия примерно на 20% выше степени сжатия для компрессора Gzip, но при этом скорость декодирования в 2-4 раза ниже. Из статьи не ясно, каким образом обеспечивалось независимое декодирование отдельных записей таблиц. Возможно, что приведенные результаты соответствуют варианту сжатия файлов целиком, без поддержки случайного доступа к записям. Таким образом, существующая реализация дает лишь незначительный выигрыш в степени сжатия над *неспециализированным* компрессором Gzip, но существенно уступает ему же в скорости декодирования. В таком виде алгоритм не может быть реальным кандидатом для организации сжатия данных в СУБД.

Сжатие табличных данных с потерями

Сжатие табличных данных с потерями информации допустимо в некоторых приложениях, например, при разведочном анализе больших массивов данных.

Идея рассмотренных в литературе методов сжатия с потерями, предлагаемых к применению в СУБД, заключается в построении модели, как можно точнее описывающей все строки. Модель задает преобразование $t_i' = F(X_i)$, позволяющее по некоторым аргументам X_i получить строку t_i', принимаемую за исходную t_i. Сжатые данные должны содержать описание F, X_i и, возможно, некоторые исходные строки t_i, отличие которых от модельных t_i' больше заданного порогового значения некоторого критерия.

Для сжатия с потерями предлагались методы кластерного анализа [47]. Под кластером[10] понимался набор строк, для которых значения некоторых атрибутов сходны. Порог сходства для объединения записей в кластеры и число сходных атрибутов задается пользователем. Поэтому правило формирование кластеров определяется как $C(k,t)$, где k — число сходных атрибутов, t — порог сходства. Для числовых атрибутов t определяет максимальную ширину диапазона значений, для категориальных — предельное число различных значений. Сжатие достигается за счет того, что для каждой строки кластера сохраняются только значений несходных атрибутов. Значения сходных атрибутов усредняются по всему кластеру. Вычисленные репрезентативные значения разделяются всеми строками кластера. Таким образом, значения k атрибутов сохраняются сразу для множества записей. С целью ускорения работы авторы алгоритма предложили строить кластеры, опираясь на центры сетки, «натянутой» со случайным шагом на множество всех возможных кластеров для конкретной таблицы. Это позволяет получить однопроходный алгоритм. Проблемой метода является задача выбора k и невозможность учета строки в нескольких кластерах, что ухудшает сжатие.

[10] Авторы алгоритма используют термин “fascicle” — «гроздь, пучок».

В [47] также рассмотрен усовершенствованный кластерный метод “ItCompress”, обеспечивающий лучшее сжатие при сходном уровне ошибки. Для каждого кластера выбирается «образцовая» запись как репрезентативный представитель. Если значение атрибута строки кластера отличается от значения атрибута образцовой записи на допустимую величину, то значение не сохраняется. Если ошибка аппроксимации велика, то такое значение-выброс запоминается. В результате сжатия для каждой строки кластера указываются: идентификатор образцовой строки кластера, битовый вектор описания значений, список значений-выбросов. Разряд битового вектора равен 1, если значение соответствующего атрибута приравнивается образцовому, и 0, если соответствующий атрибут имеет значение-выброс.

В [7] предложена схема сжатия с потерями на основе метода деревьев классификации и регрессии (Classification and Regression Tree, CaRT), названная SPARTAN. Подобно классической регрессии, метод позволяет приближенно предсказать значения некоторых атрибутов по значениям других атрибутов, играющих роль предикторов. В соответствии с предложенной схемой, если ошибка предсказания меньше задаваемого пользователем порога, то все колонки предсказываемых атрибутов целиком не сохраняются в БД. Записываются только описания моделей и колонки атрибутов-предикторов. Для ускорения процесса сжатия модели строятся для случайных выборок, а не для таблицы целиком. Недостатком подхода является используемое предположение об одинаковой силе и типе зависимостей между атрибутами в пределах всей таблицы. Если бы схема позволяла работать на уровне отдельных групп строк, это, очевидно, в общем случае увеличило бы степень сжатия при одновременном уменьшении ошибки аппроксимации.

Из [7, 47] следует, что схемы ItCompress и SPARTAN демонстрируют примерно одинаковую степень сжатия при сходной величине ошибки. На данных, использованных для сравнения, схемы обеспечивают степень сжатия порядка 5 при ошибке аппроксимации 1%. Степень сжатия аналогичных данных без потерь с помощью компрессора Gzip меньше примерно на 20%. Повышение порога допустимой ошибки до 10% увеличивает степень сжатия до 10-12 раз. При равных условиях ItCompress обеспечивает в 3-4 раза бóльшую скорость кодирования и декодирования.

В [34] обсужден вопрос эффективного сжатия с потерями баз данных, содержащих временные ряды. Считается, что исходные временные ряды сохраняются по строкам матрицы (таблицы) $\mathbf{X}$. Столбцы соответствуют моментам (интервалам) времени. Предложен метод сжатия с потерями на основе сингулярного разложения (UV-разложения, разложения на собственные числа и собственные вектора) матрицы:

$$\mathbf{X} = \mathbf{U} \times \Lambda \times \mathbf{V}^T,$$

где $\mathbf{X}$ — матрица размера $m \times n$ (m рядов, n моментов времени), $m \geq n$;

$\mathbf{U}$ — матрица ортонормированных собственных векторов матрицы $\mathbf{X}\mathbf{X}^T$, размер $m \times n$;

$\mathbf{V}$ — матрица ортонормированных собственных векторов матрицы $\mathbf{X}^T\mathbf{X}$, размер $n \times n$;

Λ — диагональная матрица собственных значений λ_i матрицы $\mathbf{X}$, размер $n \times n$.

Разложение можно записать в такой форме:

$$\mathbf{X} = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \times \mathbf{v}_i^T .$$

Исходя из свойства разложения, чем меньше номер слагаемого, тем бóльшую часть дисперсии исходных векторов, образующих матрицу $\mathbf{X}$, это слагаемое определяет. Соответственно, тем больше это слагаемое значимо. Сжатие в предлагаемой схеме достигается за счет сохранения только первых $k \le n$ слагаемых (k главных компонент). k выбирается исходя из заданного порога ошибки аппроксимации. Из приводимых в [34] результатов экспериментов следует, что степень сжатия протестированных наборов данных для предложенного метода составила 40 раз при средней ошибке в 5% и 10 раз при средней ошибке в 2%. Для сравнения, компрессор Gzip обеспечил безущербное сжатие рассмотренных таблиц в 4 раза.

Сжатие индексных структур

Основные типы индексов

В настоящее время основными типами индексных структур, используемых в РСУБД, являются B-деревья (B-tree) и битовые карты (bitmap), иначе называемые битовыми индексами [26, 28, 55]. Также применяются:

- проективные индексы (projection index) [26];
- R-деревья (R-tree) для многомерных данных [24];
- датаиндексы (DataIndex) [26].

Число же экспериментальных схем равно, видимо, нескольким десяткам [28].

Наибольшее внимание в публикациях уделяется вопросу экономного кодирования битовых карт. Интерес объясняется тем, что данный тип индексов обычно позволяет выполнять запросы существенно быстрее, чем индексы других известных типов [26, 33]. Но размер индексной структуры получается очень большим, если мощность множества значений индексируемых атрибутов велика.

В самом простом случае организации битовой карты каждой строке таблицы ставится в соответствие битовый вектор, состояние которого определяется значением индексируемого атрибута. i-ый бит принимает значение 1 для некоторой строки, если соответствующий атрибут этой строки имеет i-ое значение из множества всех значений атрибута. Например, если индексируется столбец «Пол», и область значений равна $\{мужской, женский\}$, то битовый вектор будет принимать значения:

- $\{1,0\}$ для строк со значением атрибута «Пол» = «мужской»;
- $\{0,1\}$ для строк со значением атрибута «Пол» = «женский».

Только один бит вектора может быть единичным. Длина битового вектора равна $|A|$ битам, где $|A|$ — мощность множества значений индексируемого атрибута.

Существуют другие разновидности битовых индексов [13, 55, 56]. Так называемые односторонние битовые индексы характеризуют результат одностороннего сравнения “<”, а не точного “=”. Битовые вектора в этом случае состоят из одной

последовательности единиц и одной последовательности нулей. Также разработаны битовые индексы для двухстороннего сравнения. Битовые вектора для одностороннего сравнения имеют длину $|A|-1$ битов, для двухстороннего — $|A|/2$ битов. Таким образом, проблема экономного представления индексов этих разновидностей также актуальна.

Сжатие битовых карт

В литературе обозначено два подхода к решению проблемы уменьшения размера битовых карт:

- декомпозиция, использование набора битовых карт [13, 33];
- сжатие собственно битовых карт [13, 33, 56].

Кроме уменьшения требуемого пространства для хранения, сжатие может обеспечить повышение скорости выполнения запросов. Это обуславливается следующим:

- уменьшается число операций обращения к внешней памяти;
- ряд булевых операций может быть выполнен непосредственно над сжатыми данными, причем быстрее, чем над незакодированными.

В [33] произведено сравнение четырех алгоритмов сжатия битовых карт:

1) реализация LZH в библиотеке zlib, использующей тот же формат сжатых данных Deflate, что и компрессор Gzip;
2) сжатие посредством zlib с предварительным кодированием длин серий (LZH+RLE);
3) код переменной длины, задаваемый алгоритмом ExpGol, два варианта;
4) код переменной длины, задаваемый алгоритмом байтового сжатия битовых карт (БСБК) — Byte-Aligned Bitmap Codes, два варианта [5].

Код ExpGol является расширением γ-кода Элайеса[11] для представления целых чисел. Пусть V — это набор целых чисел v_i. При кодировании n необходимо найти такое k, что

$$\sum_{i=1}^{k-1} v_i < n \le \sum_{i=1}^{k} v_i .$$

Пусть

$$d = n - 1 - \sum_{i=1}^{k-1} v_i .$$

Тогда кодовое слово для n состоит из кодового слова для k и числа d, записанного с помощью $\lceil \log_2 d \rceil$ битов. Для представления k может быть использован тот же γ-кода Элайеса: $\lfloor \log_2 k \rfloor$ нулевых битов, за которыми следует единичный бит. Например, числам $\{1, 2, 3, 4\}$ при использовании γ-кода[12] соответствуют слова

---

[11] Elias P. Universal codeword sets and representations of the integers. *IEEE Transaction on Information Theory*, 21(2):194-203, Mar. 1975.

[12] Строго говоря, в соответствии с обозначениями в исходной статье Элайеса это не γ-код, а γ'-код.

$\{1, 010, 011, 00100\}$. Для эффективного кодирования последовательности нулей используется

$V = \{1b, 2b, 4b, 8b, 16b, \ldots\}$.

Параметр b выбирается равным медиане распределения длин серий нулей для первого варианта кода ExpGol и геометрического среднему — для второго.

Код БСБК обеспечивает высокую скорость кодирования-декодирования, поскольку все действия производятся над последовательностями байтов. Булевские операции над битовыми индексами, закодированными посредством БСБК, могут выполняться существенно быстрее, чем над незакодированными.

Код БСБК может быть односторонним и двухсторонним. Односторонний код позволяет кодировать серии нулей, а двухсторонний — как нулей, так и единиц. Каждое кодовое слово одностороннего кода состоит из двух частей:

- «окно» (gap);
- окончание (ending).

Содержимое «окна» определяет, сколько нулевых байтов предшествует окончанию. Окончание может содержать последовательность незакодированных байтов или специальный контрольный байт. Используется ряд приемов для экономного кодирования размеров «окна» и окончания. Таким образом, БСБК есть разновидность байт-ориентированного кодирования длин серий.

В [33] указывается, что алгоритм LZH+RLE не продемонстрировал ни разу лучшего сжатия, поэтому его характеристики не приводятся. Было установлено, что степень сжатия ExpGol для двух вышеописанных вариантов выбора b практически не отличается. Поэтому характеристики варианта кода при вычислении b по геометрическому среднему также были опущены. В сравнении были использованы как односторонний код БСБК, так и двухсторонний. Из приводимых результатов сравнений для сгенерированных и реальных индексных данных следует:

- наибольшая степень сжатия для «густых» битовых карт с относительно большим количеством единиц обеспечивается LZH; в случае кластеризации серий нулей и единиц преимущество может иметь двухсторонний БСБК;
- наибольшую степень сжатия для разреженных битовых карт с вероятностью появления единицы $p \le 0.1$ дает ExpGol; например, при $p = 0.0001$ степень сжатия для ExpGol в три раза выше, чем у LZH;
- бóльшую скорость выполнения операций с индексами для «густых» битовых карт обеспечивает LZH;
- наибольшую скорость выполнения операций с индексами для разреженных битовых карт дает использование БСБК.

Необходимо отметить, что формат Deflate, в соответствии с которым реализована библиотека сжатия zlib, налагает ограничение на длину кодируемой за один шаг строки (длину совпадения) [1]. Этот порог равен 258 битам, что в данных условиях является жестким ограничением и неизбежно должно ухудшать степень сжатия сильно разреженных битовых карт.

В диссертационной работе [55] рассмотрены ряд проблем использования битовых индексов для поддержки сложных многомерных запросов к научным данным, содержащим большое количество недискретных атрибутов, в первую очередь

чисел с плавающей запятой. При этом изучены вопросы сжатия соответствующих модификаций битовых карт. В частности, применительно к решаемой задаче исследован вариант двухстороннего БСБК. Приводятся результаты ряда экспериментов, указывающие на ускорение выполнения запросов некоторых типов при использовании сжатых индексов.

В работах [61, 62] предложена схема сжатия на основе кодирования длин серий, обеспечивающая бóльшую скорость выполнения операций с индексами, чем БСБК. Метод назван авторами «пословное гибридное кодирование длин серий» (Word-Aligned Hybrid run-length code, WAH). Название определено тем, что закодированные данные группируются в машинные слова, а не в байты, как в БСБК. Соответственно, вся обработка может выполняться на уровне слов. Определены два типа слов: заполнители (“fill words”) и литералы. Слова различаются значением старшего бита. В слове-заполнителе указывается значение бита, образующего серию, и длина серии таких одинаковых значений. В литералах сохраняются в исходном виде те биты, которые не удалось сжать с помощью заполнителей. Требуется, чтобы длина серии, кодируемой заполнителем, была кратной числу рабочих битов литерала. Например, если машинное слово состоит из 32 битов, то длины совпадений должны быть кратны 31.

Результаты сравнительных экспериментов, приведенные в [62], свидетельствуют о 12-кратном превосходстве в скорости выполнения запросов при использовании предложенной схемы пословного кодирования относительно сжатия с помощью БСБК. Размер индексной структуры при этом на 60% больше размера структуры для БСБК. Для тестирования была использована таблица с данными результатов физических экспериментов, содержащая порядка 2.2 миллиона записей. Отмечается, что при вероятности появления в битовой карте единицы $p = 0.01 .. 0.001$ размер индекса для предложенного метода примерно в 2.5 раза превышает размер индекса для БСБК.

Следует отметить, что основной проблемой всех способов сжатия битовых индексов, основанных на кодировании длин серий, является решение задачи эффективного декодирования с произвольным доступом.

Экономное кодирование B-деревьев

Сжатие индексных данных в B-дереве позволяет не только уменьшить размер структуры, но и приводит к получению более «плоского» дерева, т.е. с меньшей высотой. Это ускоряет доступ к отдельному ключу.

Обеспечение возможности непосредственного сравнения сжатых данных требует использования алгоритмов кодирования с сохранением упорядоченности. В [24] указано, что для сжатия вершин B-дерева может быть использован предложенный вариант метода упаковки битов, названный «сжатие кадра ссылок». Эта модификация была рассмотрена выше в подпункте «Упаковка битов» пункта «Кодирование числовых данных». Как и обычный метод упаковки битов, данный вариант сохраняет порядок сортировки, но годен только для кодирования целых чисел. Универсальная схема сжатия с сохранением упорядоченности описана в работе [6]. Алгоритм является словарным и в этом сходен с предложенной в [63] схемой модификации LZW для сохранения упорядоченности. Но он обеспечивает

бóльшую степень сжатия. В статье указано, что предложенный алгоритм позволил сжать реальные индексные данные в 5 раз.

Сжатие индексов других типов

Вопрос экономного кодирования R-деревьев обсуждается в [24, 25]. Для сжатия таких структур предлагается использовать схему «сжатие кадра ссылок», основанную на методе упаковки битов и рассмотренную выше в подпункте «Упаковка битов» пункта «Кодирование числовых данных».

Схемы сжатия датаиндексов изучены в [26, 27]. Экономное кодирование индексных данных выполнялось на уровне страницы. Наилучшие результаты с точки зрения степени сжатия были получены для LZW с предварительным взятием разности значений (дифференциальным кодированием).

Сжатие результирующих наборов

Сжатие результирующих наборов, получаемых в итоге выполнения запросов, актуально в силу ряда причин:

- уменьшается время передачи результата клиенту, что особенно значимо при низкой пропускной способности канала связи (например, радиоканала);
- экономится память клиентского вычислительного устройства, которая в случае карманных компьютеров может иметь малый объем и являться критическим ресурсом;
- уменьшается время обмена информацией при выполнении запроса к распределенной системе, когда запрос разбивается на подзапросы, выполняемые на различных компьютерах.

Как уже отмечалось, вопросу сжатия результирующих наборов в литературе уделяется недостаточно внимания.

В [14, 16] предложена общая схема сжатия результирующих наборов, учитывающая вопросы выбора совокупности методов сжатия для кодирования набора на основании статистической и семантической информации. Последнее включает знание ограничений целостности в БД и запроса как такового. Например, учет упорядоченности по некоторым столбцам.

Введено понятие «плана сжатия» результата (“compression plan”), состоящего из совокупности «операторов сжатия». Каждый оператор сжатия соответствует применению определенного алгоритма экономного кодирования к фрагменту результирующего набора. В общем случае фрагмент уже может быть сжат за счет использования нескольких операторов. Для упрощения анализа в [14] фрагмент принят равным одному или нескольким столбцам. Оператор сжатия O определяется как $O = (A_O, G_O, E_O, m_O, t_{in}, t_{out})$, где A_O — набор сжимаемых столбцов, G_O — входная гранулярность, определяющая минимальный блок данных, который может быть сжат посредством O, E_O — выходная гранулярность, m_O — алгоритм сжатия, t_{in} — требуемый тип входных данных, t_{out} — тип выходных данных. План сжатия p определен как $p = O_N(O_{N-1}(\ldots(O_1(R))\ldots))$, где R — сжимаемый набор данных. В общем случае изменение порядка применения операторов дает иной

план. В зависимости от типа операторов и их взаимозависимостей план может выполняться в параллельном и конвейерном режимах. Например, параллельно могут применяться операторы, кодирующие разные колонки таблицы.

Эффективность экономного кодирования определяется выбором плана сжатия, оптимального или достаточно хорошего по заданному критерию. В [14] использован следующий вид критерия стоимости плана:

$$C(p) = w_1 \cdot c_c + w_2 \cdot c_d - w_3 \cdot s_n - w_4 \cdot s_s,$$

где w_i — перенастраиваемые константы, задающие значимость соответствующих затрат;

c_c — цена кодирования;

c_d — цена декодирования;

s_n — экономия пропускной способности канала связи;

s_s — экономия памяти клиентского вычислительного устройства.

При поиске удовлетворительного плана сжатия также должно обеспечиваться согласование типа и гранулярности входных и выходных данных операторов. Например, оператор, применяемый к значению атрибута, не может быть использован после оператора, сжимающего на уровне строки. Кроме того, налагается ограничение, что каждый оператор может только один раз войти в план.

Исчерпывающий алгоритм поиска наилучшего плана, состоящий в переборе всех вариантов, является конечным, поскольку конечно множество операторов, и каждый из них может быть использован только один раз. Но временна́я сложность алгоритма экспоненциально зависит от количества сжимаемых фрагментов. Если, как принято в работе, фрагмент состоит целиком из одного или нескольких столбцов, то это количество столбцов. Поэтому предложен практически ценный субоптимальный эвристический алгоритм, временная сложность которого на практике близка к $O(a)$, где a — количество атрибутов в наборе.

Для проведения экспериментов в работе были использованы несколько алгоритмов:

- нормализация и группировка отсортированных данных;
- упаковка битов через кодирование разницы между текущим значением атрибута и нижней границей его диапазона значений;
- кодирование по полуадаптивному словарю значений атрибута;
- дифференциальное кодирование;
- LZH-сжатие (применялся компрессор WinZip);
- устранение ведущих нулей чисел, а также ведущих и оконечных пробелов строк;
- кодирование по Хаффману;
- арифметическое сжатие.

Схемы нормализации и группировки отсортированных данных предложены в этой же работе. Нормализация отсортированных данных предполагает устранение из набора значений атрибутов, которые находятся в известной функциональной зависимости от известных значений других атрибутов. Например, пусть результирующий набор был получен слиянием двух таблиц R_1 и R_2 с атрибутами (A, B) и

(C, A) соответственно, при этом A является первичным ключом в R_1. Тогда можно определить B, зная A. Если результат отсортирован по A, то набор на Рис. 4 может быть без потери информации представлен в виде Рис. 5. Схема группировки отсортированных данных аналогична, но позволяет устранить дублирование значений только тех столбцов, по которым отсортирован набор.

A	*B*	*C*
1	2	7
1	2	4
3	3	9

Рис. 4. Результирующий набор без сжатия

A	*B*	*C*	Признак нормализации
1	2	7	1
		4	0
3	3	9	1

Рис. 5. Результирующий набор после нормализации отсортированных данных

Выбор алгоритма при формировании плана сжатия определялся статистической и семантической информацией, известной после оптимизации плана выполнения запроса (т.е. до собственно выполнения запроса). Результаты экспериментов, проведенные с использованием запросов теста TPC-D, свидетельствуют о возможности многократно — в 2 и более раз — улучшить сжатие за счет выбора хорошего плана сжатия, учитывающего семантическую информацию о данных. Также отмечается ускорение передачи данных с учетом времени кодирования и декодирования, если пропускная способность канала составляет менее 500 кБайт/с, а клиентское устройство представляет собой типовой персональный компьютер. Если клиентское вычислительное устройство является «карманным» компьютером, то это требует применения более простых методов сжатия, с тем чтобы декодирование выполнялось с меньшими затратами процессорного времени и памяти. Указывается, что применение сжатия результирующих наборов в этом случае позволило в 2 раза уменьшить расход памяти клиентского «карманного» компьютера при замедлении доступа к данным только на 15%.

Исследования, описанные в [14], при всей своей ценности не исчерпывают вопрос сжатия результирующих наборов. Он требует более полной и детальной проработки.

Сходные области исследований: сжатие данных в информационно-поисковых системах

Информационно-поисковые системы основаны на использовании текстовых баз данных. Создание таких систем связано типичными проблемами практической реализуемости и производительности, разрешение которых требует применения методов экономного кодирования данных.

Информационно-поисковая система должна в ответ на текстовый запрос выдавать набор документов (текстов), отсортированный по уменьшению предполагаемого соответствия документа запросу. Обычно запрос представляет собой совокупность слов, которые должны встречаться в требуемом тексте.

Типовая схема реализации хранения данных предполагает экономное представление как собственно документов, так и индексных структур, обеспечивающих быстрый анализ соответствия документа и доступ к нему [9, 38, 39]. В качестве основной схемы индексирования используется так называемый словоуказатель, или инвертированный файл (inverted file). Словоуказатель позволяет найти для заданного слова все тексты, в которых оно встречается. Каждая запись словоуказателя соответствует одному слову из всего множество слов, употребляющихся в индексируемых документах. Запись состоит из набора идентификаторов текстов, в которых встречается слово, и частот встречаемости. Запись также включает ссылку на слово в словаре уникальных слов или само слово. Словарь может представляться в виде обычного массива или некоторой структуры с индексированием, например B-дерева [39]. Отображение идентификаторов документов в их физические адреса выполняется с помощью таблицы. Задача экономного представления словоуказателя аналогична проблеме сжатия индексных структур в обычных СУБД. Для кодирования словоуказателя применялись как методы сжатия битовых карт, например код Голомба [38], так и методы кодирования целых чисел, в частности коды Элайеса [9].

Сжатие собственно текстов исследователи предлагали выполнять путем полуадаптивного кодирования на уровне слов. В этом случае общая для всей БД модель дает возможность независимого декодирования отдельного текста, и обеспечивается бóльшая скорость декодирования, чем при посимвольном сжатии. Например, в [38] для сжатия текстов предложен полуадаптивный алгоритм Хаффмана.

Таким образом, вопросы применения сжатия данных в информационно-поисковых системах аналогичны возникающим при экономном кодировании баз данных произвольного вида, содержащих редко изменяющуюся, статическую информацию.

Оптимизация запросов в СУБД со сжатием данных

Использование сжатия данных требует соответствующей переработки подсистемы оптимизации плана выполнения запросов и, возможно, изменения алгоритмов оптимизации. Это объясняется высокой вычислительной стоимостью операций декодирования и кодирования.

В работах [14, 15] произведен анализ проблемы и предложен вариант решения. Рассмотрены несколько стратегий выполнения декодирования при отработке запросов к БД.

1. «Жадное» декодирование, при котором данные разжимаются при их передаче в основную память. «Жадное» декодирование предлагается в ранних работах по сжатию в СУБД, например, в [32]. Преимущество жадного декодирования в локализации изменения программного кода, обеспечивающего работу со сжатой БД, на уровне модуля управления памятью. Такая стратегия является естественной, если предполагается аппаратная реализация сжатия. Но при «жадном» декодировании генерируются субопти-

мальные планы выполнения запросов, поскольку не учитывается возможность выполнения непосредственно над сжатыми данными таких операций, как, например, проецирование и объединение по эквивалентности [29]. На практике таблицы БД уровня предприятия часто содержат сотни колонок, но большинство запросов задействуют всего несколько. «Жадное» декодирование в таких случаях крайне неэффективно.

2. «Ленивое» декодирование предполагает, что данные остаются сжатыми в течение выполнения запроса до тех пор, пока это допустимо, а декодируются только при необходимости, после чего остаются разжатыми на выходе. Такой подход является более эффективным. Однако использование этой стратегии может привести к увеличению размера промежуточных наборов, поскольку данные остаются разжатыми на выходе. В результате может увеличиться число операций ввода-вывода при выполнении последующих действий плана запроса.
3. «Вре́менное» (transient) декодирование, предложенное в [14], позволяет избежать увеличения размеров промежуточных результатов. При использовании данной стратегии реляционные операторы модифицируются так, чтобы временно разжимать данные, если это необходимо для выполнения операции, но оставлять их на выходе сжатыми. Такие операторы, получающие и возвращающие сжатые данные, автор подхода называет «вре́менными» операторами. При использовании временного оператора данные, находящиеся в буферном пуле, всегда остаются сжатыми. Разжатое значение сохраняется во временной программной переменной и «забывается» сразу после использования.

В рамках реализации стратегии «временного» декодирования в [14] предложены методы нахождения оптимального и субоптимального плана выполнения запроса в СУБД со сжатием данным. Дана оценка сложности алгоритма определения оптимального плана. Сложность по машинной памяти составляет $O\left(2^n \cdot 2^k\right)$, где n — число отношений в запросе, k — число атрибутов, требующих декодирования (либо для выполнения операций, либо для формирования результирующего набора). Сложность по времени равна $O\left(n \cdot 2^{n-1} \cdot 3^k\right)$. При этом для типового оптимизатора сложность по машинной памяти составляет $O\left(2^n\right)$, сложность по времени — $O\left(n \cdot 2^{n-1}\right)$. Для предложенного эвристического субоптимального алгоритма поиска, обеспечивающего высокую скорость обработки, сложность по машинной памяти равна $O\left(2^n \cdot K\right)$, где K — наибольшее число анализируемых планов с наименьшей ценой выполнения, сложность по времени — $O\left(n \cdot 2^{n-1} \cdot 2K\right)$. Экспериментально показана эффективность решений. План, выбираемый по субоптимальному алгоритму, для всех тестов совпал с оптимальным при $K \geq 2$. Использование субоптимального алгоритма в сочетании с «временным» декодированием позволило в 2-10 раз увеличить скорость выполнения запросов и сэкономить до 50% от объема буферного пула.

В работе [4] предлагается алгоритм оптимизации плана вычисления булевских выражений, выполняемых с использованием сжатых битовых индексов. Из результатов расчетов могут быть определены наилучший формат представления битового индекса и способ выполнения заданной булевской операции. Алгоритм

основан на динамическом программировании и имеет линейную сложность по времени. Используется модель стоимости выполнения операций на основе оценок разреженности битовой карты и скорости выполнения действий для конкретного формата представления битового индекса. Отмечается увеличение скорости выполнения запросов на десятки процентов вплоть до нескольких раз в случае использования схем сжатия битовых индексов БСБК и ExpGol. Максимальные улучшения зафиксированы в тех случаях, когда индексированные атрибуты имеют высокую мощность множества значений.

Проблема оптимизации плана выполнения запроса при использовании экономного кодирования также затронута в [59]. Предложен достаточно простой вариант модификации стандартной подсистемы оптимизации посредством:

- добавления оценок стоимости декодирования полей записи в модель стоимости операций;
- изменения оценок стоимости операций обмена с оперативной памятью и внешней памятью, поскольку они ниже при использовании сжатия; предлагается эмпирический коэффициент пересчета 0.5;
- обеспечения возможности сжатия промежуточных результатов, если они сохраняются во внешней памяти.

Результаты исследований показывают, что задача разработки адекватной системы оптимизации плана выполнения запроса является ключевой для СУБД со сжатием данных.

Выводы

Методы сжатия, предлагаемые к использованию в СУБД, обычно представляют собой разновидности кодирования длин серий и словарного сжатия, обеспечивающие быстрое декодирование. Во многих схемах учет сильных вертикальных связей между значениями колонок учитывается с помощью предварительного дифференциального кодирования. Практически не применяются сложные статистические методы. Слабо исследованы возможности автоматического распознавания и учета структуры данных. Сходные подходы применяются только для сжатия с потерями информации.

Обычно приводимые оценки степени сжатия баз данных составляют 2..5 раз, часто отмечается ускорение выполнения запросов за счет сжатия на несколько десятков процентов и больше. Но объективное сравнение различных способов экономного кодирования затруднено, поскольку нечасто приводятся характеристики реализаций для стандартных тестовых баз данных. Отсутствие общепризнанных эталонов является большой проблемой само по себе.

В области экономного кодирования индексов основной интерес для исследователей имеет поиск формата битовых индексов, который бы обеспечил наибольшую производительность при выполнении запросов.

Задача экономного представления результатов набора практически не рассматривается в литературе.

Проблеме оптимизации планов выполнения запросов при использовании сжатия только в последнее время стало уделяться должное внимание. Известные результаты свидетельствуют о возможности повышения производительности СУБД

на десятки процентов и более за счет изменения оптимизатора и алгоритма выполнения реляционных операторов.

Также следует отметить недостаточную активность в создании и внедрении методов сжатия, позволяющих оперировать при выполнении запросов непосредственно закодированными данными. Существующие разработки ориентированы на кодирование индексных данных.

2. Использование сжатия данных в современных СУБД

Сжатие данных поддерживается практически во всех современных СУБД. Ниже дан обзор приемов экономного кодирования, применяемых в конкретных системах. Описания упорядочены в соответствии с алфавитным порядком наименований СУБД[13].

ADABAS

СУБД ADABAS немецкой фирмы Software AG является системой уровня предприятия, предназначенной для использования в сложных приложениях с большим количеством активно работающих пользователей и жесткими требованиями к производительности и надежности.

Система ADABAS поддерживает сжатие таблиц базы данных [53]. В версии 6.2 реализованы два способа сжатия, работающих на уровне значений атрибутов:

1) сжатие по умолчанию, соответствующее использованию полей переменной длины, при котором из строковых значений устраняются завершающие пробелы, а из числовых значений удаляются ведущие нули;
2) сжатие отсутствующих значений; в этом случае в дополнение к использованию полей переменной длины экономно представляются последовательности отсутствующих значений, принадлежащих одной записи; таким образом может быть одновременно закодировано до 63 отсутствующих значений атрибутов, примыкающих друг к другу.

В [53] указывается, что встроенные методы сжатия позволяют сократить размер базы примерно в 2 раза относительно формы представления с полями фиксированной длины.

В версии 7.1.2 СУБД последовательности отсутствующих значений в конце строк полностью отсекаются и физически не сохраняются [31]. В этой версии используется и сжатие индексов, при котором устраняется повторение начальной части индексных значений, располагаемых друг за другом в индексной структуре.

[13] Отсутствие в обзоре данных об Microsoft SQL Server — одном из основных продуктов на рынке СУБД — объясняется не антипатиями автора, как некоторые могли подумать, а отсутствием какой-либо информации об использовании экономного кодирования данных в этой системе.

DB2

Системы управления данными фирмы IBM традиционно содержат механизмы экономного кодирования информации. СУБД DB2 поддерживает сжатие данных как на программном уровне, так и на аппаратном [10, 32].

В версиях DB2 ниже 3-ей таблицы могли быть сжаты с помощью специальных процедур, ассоциируемых с таблицей при ее создании. В частности, предоставлялся исходный код типовой реализации алгоритма Хаффмана. Пользователи могли модифицировать алгоритм или создавать свои процедуры, реализующие иные алгоритмы. Ряд независимых производителей предлагал набор такого рода модулей.

Начиная с версии 3 DB2 поддерживает словарный метод сжатия, реализованный в аппаратной платформе S/390. В последнем случае кодирование-декодирование выполняется специальным процессорным модулем мейнфрейм-платформы S/390. Если процессорный модуль сжатия не доступен, то используется программная реализация. Алгоритм сжатия основан на одном из вариантов LZW, известном как метод Миллера-Уэгнама (Miller-Wegnam), или LZMW [32, 1]. В отличие от LZW, в методе Миллера-Уэгнама каждая фраза словаря представляет собой конкатенацию двух других фраз, а не фразы и символа алфавита данных. В DB2 сжатие выполняется на уровне строки и является полуадаптивным. Словарь строится при загрузке данных. В случае необходимости можно собрать статистику и реорганизовать словарь.

Сжатию может подвергаться как таблица, так и ее отдельные разделы. Соответственно, каждая сжатая таблица или раздел таблицы имеют свой словарь, размещаемый в заголовке объекта как метаинформация. Не все записи могут подвергаться кодированию. Если определяется, что для некоторой записи процедура сжатия не дает положительного эффекта, то запись помечается соответствующим флагом и сохраняется в исходном виде. Также записи могут не кодироваться, если это приводит к получению более 255 строк на страницу, что обусловлено ограничениями внутреннего устройства СУБД.

Словарь может включать в себя 512, 1024, 2048 и 4096 фраз. Объем словаря в байтах равен размеру во фразах, умноженному на 16.

В [32] отмечается, что выбор способа сжатия был обусловлен следующими аргументами:

- большинство колонок имеет небольшую длину, поэтому сжатие на уровне значений атрибутов неэффективно в первую очередь за счет необходимости указывать длину каждого закодированного значения; в результате в качестве сжимаемого блока выбрана строка;
- адаптивное сжатие строк без предварительных знаний не дает существенной экономии и проигрывает статическому, поскольку часто строки имеют длину в несколько десятков или несколько сотен байт и содержат неоднородную информацию;
- словарная схема типа LZW обеспечивает значительно более быстрое декодирование, чем арифметическое сжатие и кодирование по Хаффману с различными статистическими моделями, при одинаковой или большей степени сжатия типовых табличных данных.

Построение словаря и сжатие может выполняться двумя способами с помощью двух различных утилит, LOAD и REORG. Утилита LOAD обеспечивает загрузку данных в таблицу. При этом записи, использованные для построения словаря, не сжимаются. Сжатие строк выполняется только после заполнения словаря. Это может заметно ухудшать сжатие для небольших таблиц. REORG обеспечивает экономное кодирование всех строк. В REORG выполняется построение словаря на основании выборки строк из всего массива данных сжимаемого объекта. При формировании выборки предпочтение отдается первым по порядку записям: чем дальше расположены записи от начала таблицы (раздела), тем с б*о*льшим шагом они выбираются. Для улучшения сжатия сначала строится словарь размером вплоть до $2^{14} = 16384$ фраз, что позволяет накопить более полную статистику, а затем уменьшается до допустимого объема удалением редко используемых элементов.

Утилита REORG используется и для перестройки словаря в случае необходимости. Например, после существенного изменения содержимого таблицы.

В буферном пуле страницы сохраняются сжатыми. Декодируются только выбираемые по запросу записи. При вставке и обновлении записей они сжимаются и записываются в буферный пул. Это позволяет уменьшить расход памяти под буферный пул и/или увеличить процент успешных попаданий в него.

Записи журнала (регистрационного файла СУБД), соответствующие сжатым строкам, также содержат данные в закодированном виде.

Следует отметить, что в DB2 версий 3-5 нет возможности сжимать индексы.

В [10] и [32] приводятся следующие результаты сравнения характеристик системы в зависимости от использования сжатия:

- процент сэкономленного пространства при сжатии с помощью утилиты LOAD равен 41% и 56% для данных с умеренной и высокой избыточностью соответственно, что эквивалентно степени сжатия в 2 и 2.3 раза;
- загрузка данных в таблицы посредством LOAD выполняется на 30% медленнее;
- сканирование таблиц выполняется в целом примерно в 2 раза быстрее, но при этом нагрузка на процессоры возрастает также двое (было задействовано программное кодирование-декодирование); показатели для случайного доступа практически не отличаются от соответствующих показателей для последовательного доступа;
- быстрота создания резервных копий и восстановления после сбоев также увеличилась в 2 раза.

Ingres

Система Ingres фирмы Computer Associates представляет собой промышленную СУБД, ориентированную на приложения уровня предприятия.

Следует отметить, что существуют две разновидности Ingres: свободно распространяемая «академическая» Ingres и коммерческая, разрабатываемая и продаваемая Computer Associates. «Академическая» Ingres (INteractive Graphics REtrieval System — интерактивная система доставки графических данных) является исходной версией системы, разработанной в Университете Беркли в 1970-х годах для демонстрации возможностей РСУБД. Поэтому есть серьезные основания считать In-

gres первой реализацией РСУБД, ведь она была создана раньше канонического комплекса System R фирмы IBM. Коммерческая Ingres создавалась на основе «академической», но в настоящее время это программы с принципиально разными характеристиками. Вся последующая информация касается исключительно коммерческой Ingres фирмы Computer Associates.

Текущие версии Ingres выпускаются под маркой Advantage Ingres. Предыдущими марками являлись наименования OpenIngres и Ingress II.

Источники, использованные при подготовке данного обзора, противоречивы относительно методов сжатия, применяемых в Ingres.

Из руководства администратору для версии 2.6 системы следует, что структуры, в которых размещаются данные, могут сжиматься за счет использования строк переменной длины при хранении *текстовых* полей и экономного кодирования отсутствующих значений (NULL) атрибутов любого типа [19]. При этом представляются в сжатом виде как собственно данные, так и индексная информация. В зависимости от типа структуры данных, может не поддерживаться сжатие первичных ключей и индексов. Таким образом, способы сжатия являются тривиальными.

В неформальном тексте “Ответы на часто задаваемые вопросы по Ingres” [30] указывается, что в версиях СУБД OpenIngres 1.x использовался метод сжатия LZW. Автор текста [30] не имеет непосредственного отношения к фирме Computer Associates, поэтому данную информацию нельзя считать заслуживающей доверия. Возможно, что метод LZW действительно применялся в ранних версиях, но не используется в современной Advantage Ingres.

Microsoft Access

Системообразующим компонентом «настольной» СУБД Microsoft Access является программа Microsoft Jet, обеспечивающая чтение и сохранение информации в базах данных. Начиная с версии 3 Microsoft Jet обеспечивается сжатие ключей в индексах [18]. Исходя из имеющейся информации тип метода сжатия не представляется возможным определить. В источнике лишь указывается, что в результате экономного кодирования устраняется дублирование значений ключей.

MySQL

MySQL относится к группе простых систем управления данными. Но в последние годы система получила большую популярность. Значительную роль в этом сыграли статус программы с открытым исходным кодом и простые интерфейсы взаимодействия с программами на языках программирования PHP, C, C++, Java и ряду других, что позволило без больших затрат встраивать MySQL в прикладные системы. Отсутствие в MySQL большого количества системных и вспомогательных функций, характерных для промышленных РСУБД, компенсируется высокой скоростью выполнения запросов на чтение. MySQL распространяется разработчиком, фирмой MySQL AB, под двумя лицензиями: "общедоступной" лицензии для открытого ПО General Public License (GPL) или альтернативной коммерческой лицензией. Базовая лицензия GPL диктует, в частности, необходимость распространения с исходным кодом как MySQL, так и непосредственно использующих ее программ.

В версии 4.0.18 использование сжатия данных представлено в следующих аспектах [41].

1. Во-первых, имеется возможность создавать сжатые таблицы только для чтения. При этом создаются таблицы с индексно-последовательным методом доступа (ISAM), поэтому при использовании индексирования соответствующие структуры тоже содержат сжатые данные. Создание таких таблиц выполняется посредством утилиты Myisampack. Сжатие выполняется на уровне значения атрибута. Утилита позволяет экономно закодировать таблицу с помощью:

- упаковки битов;
- устранения колонок-констант, в том числе состоящих из неопределенных (NULL) значений;
- устранения ведущих и оконечных пробелов;
- устранения ведущих нулей;
- словаря значений атрибута;
- кодирования по Хаффману.

Методы могут применяться совместно, в том числе допускается каскадное использование. Таблицы кодов Хаффмана могут использоваться для декодирования сразу нескольких колонок. При сжатии таблицы генерируется статистика, которая используется оптимизатором запросов. Сжатая таблица может быть преобразована в обычную.

2. Во-вторых, экономно представляются индексные структуры типа B-дерева. Одинаковые префиксы значений и оконечные пробелы в B-дереве устраняются. Для улучшения сжатия ключи числовых форматов могут записываться старшими байтами вперед.

Кроме того, в версии 4.1.1 поддерживается кодирование-декодирование значения столбца с помощью функций Compress/Uncompress:

SELECT Uncompress(Compress("any string")); → "any string"

Использование этих функций требует, чтобы система MySQL была скомпилирована с библиотекой сжатия zlib (реализует LZH) или подобными.

Oracle

РСУБД Oracle одноименной фирмы является наиболее распространенной промышленной системой управления данными. В настоящее время последней версией комплекса является Oracle9*i* Release 2.

В СУБД Oracle реализовано как сжатие таблиц, так и сжатие индексов. Но если экономное представление индексов использовалось еще в Oracle версий 7 и 8 [13], то полноценное сжатие таблиц поддерживается только в самой последней версии [46].

В Oracle версий 7 и 8 существовал особый вид структуры хранения данных под названием «индексная таблица» (index-organized table). Данные такой таблицы сохранялись совместно с индексной информацией (атрибуты первичного ключа такой таблицы описывались только в индексной структуре) [45]. При этом поддерживалось частичное сжатие значений составного первичного ключа в индексе. Значение составного ключа разделялось на две части: группирующая (префикс), общая для нескольких значений, и уникальная (суффикс). Префикс подвергался эконом-

ному кодированию. Возможно, при этом используется словарный алгоритм с сохранением упорядоченности, описанный одним из сотрудников корпорации Oracle в [6].

Табличное сжатие в Oracle9*i* Release 2 в первую очередь предназначено к использованию в системах поддержки принятия решений и OLAP-приложениях[14]. Сжатию могут быть подвергнуты целые таблицы, разделы таблиц и материализованные представления. Сжатие уменьшает используемый объем дискового пространства, снижает требования к размеру буфера данных и во многих случаях ускоряет обработку запросов, особенно для систем с медленным вводом-выводом информации.

Для сжатия таблиц используется метод словарного типа [46]. Сжимается или целиком столбец, или набор столбцов. Словарь формируется из значений атрибутов. Решение о добавлении значения столбца в словарь принимается на основании длины столбца и числа повторов значения. Короткие, а также редко встречающиеся значения в словарь не вносятся. Сжатие производится путем замены значения столбца на указатель на соответствующую фразу словаря. Размер словаря выбирается автоматически для каждой страницы во время загрузки данных. При этом таблица фраз и другая необходимая для декодирования информация сохраняется в каждой странице.

Для повышения степени сжатия может производиться переупорядочивание столбцов внутри одной страницы. Данное преобразование полностью скрыто от внешней среды. При значительных изменениях в данных словарь адаптируется.

В [46] утверждается, что внедрение алгоритма сжатия потребовало изменения только тех модулей, которые связаны с формированием блока и доступом к строкам и столбцам. Нет никакой разницы в функциональности, обеспечиваемой сжатыми и несжатыми таблицами.

Достигаемая степень сжатия и изменение скорости выполнения запросов зависят от характера данных и вида обращений к БД. В [46] приведены результаты тестирования для двух схем: типовое хранилище данных, организованное по схеме «звезда» и имеющее 1 таблицу фактов и 5 таблиц с измерениями, и стандартная схема из тестов TPC-H/TPC-R, ориентированных на оценку СППР. Последняя схема содержит 8 таблиц и является нормализованной, хотя ее структура не относится к типу «звезда». Отмечаются следующие результаты:

- для первой схемы уменьшение размера базы составило 3.1 раза, при этом скорость выполнения типовых сложных запросов увеличилась на 13%;
- для второй схемы степень сжатия составила 1.4 раза, а общее повышение производительности равнялось 10%.

Первоначальный размер первой базы составлял 55 Гбайт, второй — 115 Гбайт. Для первого примера сжималась только таблица фактов и материализо-

[14] OLAP — On-Line Analytical Processing — технология обработки многомерной информации в реальном времени, связанная с динамическим составлением интерактивных отчетов табличного и графического вида. Интерактивные отчеты представляют сводные сведения о массиве многомерной информации, полученные путем агрегирования данных по каким-то измерениям или иерархическим уровням измерений массива информации, и, как минимум, позволяют динамически изменять уровень агрегации. Эффективная поддержка OLAP требует особого устройства базы данных, поэтому OLAP обычно используется совместно с многомерными базами данных или РБД, организованными по принципам хранилищ данных (data warehouse).

ванные представления, поскольку размер таблиц с измерениями обычно мал для схем типа «звезда». Для второго примера сжимались две самые большие таблицы. Подобно таблице фактов в первой схеме, эти таблицы содержали числовые данные. Разница в сжатии обусловлена в первую очередь большей мощностью множеств значений, принимаемых атрибутами таблиц во втором примере. Кроме того, распределения значений колонок для второй схемы более равномерны, что ухудшает сжатие.

Битовые индексы представляются в базах Oracle в экономном виде. В [13] указывается, что в Oracle версии 8 используется схема сжатия битовых индексов, описанная в патенте [5]. Этот метод сжатия, байтовое сжатие битовых карт, позволяет корректно сравнивать данные без их декодирования. Метод кратко описан выше в пункте «Сжатие битовых карт» параграфа «Сжатие индексных структур».

В [37] утверждается, что в версии Oracle 8*i* Release 8.1.5 поддерживается частичное сжатие значений составного первичного ключа в индексе типа B-дерева, аналогичное экономному кодированию первичного ключа в индексных таблицах.

SAS System

Программный комплекс SAS System фирмы SAS Institute не является СУБД как таковой. Это сложная система для хранения, анализа и доставки информации. Хотя некоторые специалисты причисляют SAS System к системам управления статистическими базами данных [54], данный программный комплекс не обеспечивает транзакционную работу и имеет сравнительно скромную поддержку совместного доступа к данным на запись. Тем не менее, SAS System часто применяется при организации корпоративных хранилищ данных и создании OLAP-приложений.

В SAS System реализовано 2 метода сжатия данных, и при этом есть возможность программировать иные алгоритмы сжатия с помощью специального инструментального средства разработки [52]. Первым методом является кодированием длин серий (RLE). Эта техника очень эффективна для кодирования таблиц с текстовыми колонками, поскольку по умолчанию все строки любой таблицы в SAS System имеют одинаковую фиксированную длину. Вторым методом является метод сжатия Росса (Ross Data Compression)[15]. В этой технике применяется словарное сжатие со скользящим окном в сочетании с кодированием длин серий. Данные сжимаются по записям, т.е. единицей сжатия является строка. Метод ориентирован на сжатие числовых полей и хорошо работает при длине строки от нескольких сотен байтов и больше.

Сжатие всегда применяется только ко всей таблице. Способ сжатия можно задавать только для создаваемой таблицы. При этом также можно указать два способа удаления и вставки новых записей в сжатую таблицу. При первом способе строки вставляются в конец таблицы вне зависимости от наличия вакантных секций (слотов) внутри таблицы. При втором способе отслеживаются пустующие секции внутри таблицы и вставляемые записи могут быть размещены в них, что при-

[15] Судя по описанию в [52], это действительно алгоритм Э. Росса, опубликованный с исходным кодом реализации в журнале "The C Users Journal": E. Ross. A Simple Data Compression Technique. *The C Users Journal*, 10(10):113-120, Oct. 1992.

водит к экономии дискового пространства. Но в последнем случае нельзя обратиться к строке по ее номеру.

Также необходимо отметить возможность организации словарного сжатия значений атрибутов. Это обеспечивается механизмом форматов, позволяющим поразному представлять данные. Можно задать такой формат, который поставит в соответствие числам строковые константы. Например, можно определить формат $regions., в соответствии с которым при обработке данных будут автоматически выполняться преобразования типа: 1 → "Европа", 2 → "Азия" и т.п. Если сопоставить такой формат числовой колонке таблицы, то получится реализация словарного сжатия на уровне значений атрибута. В общем случае использование форматов требует определенного учета в прикладных программах, т.е. процесс работы с преобразованными данными не является «прозрачным».

Sybase IQ

Sybase IQ — это высокопроизводительная СУБД, ориентированная на использование в системах поддержки принятия решений.

В [24] со ссылкой на одного из сотрудников фирмы Sybase указывается, что в Sybase IQ используется схема сжатия на основе LZ76[16]. При этом не поддерживается избирательное декодирование отдельных кортежей, сжимается целиком страница. В буферном пуле сохраняются как распакованные, так и сжатые версии страниц. В работе [25] того же автора утверждается, что схема сжатия в Sybase IQ сходна с алгоритмом сжатия, используемым в компрессоре Gzip. Указывается, что применяется коммерческая версия Gzip, разработанная фирмой Stacker. Поскольку Gzip использует формат сжатых данных Deflate [1] и, следовательно, реализует метод LZH, то можно сделать заключение об использовании в Sybase IQ словарной схемы типа LZ77 с последующим блочно-адаптивным кодированием по алгоритму Хаффмана.

Из руководства администратору СУБД [57], относящегося к Sybase IQ версий 11.2.x, следует, что в результате сжатия физические страницы (таблиц базы) размером 16 условных блоков преобразуются в логические. При стандартном размере блока в 4 кБайт размер физической страницы составляет 64 кБайт. Размер логической страницы может составлять 1, 2, 4, 8 и 16 блоков. Логические страницы размещаются последовательно во внешней памяти и считываются порциями, соответствующими размеру физической страницы. Таким образом, вне зависимости от характеристик данных и размера блока, максимальная степень сжатия равна 16. Размер физической страницы может быть уменьшен с используемых по умолчанию 16 блоков до 2, 4 или 8 блоков. В этих случаях максимальная степень сжатия уменьшается до 2, 4 и 8 соответственно, но требуется меньше оперативной памяти для буферов и выполнения декодирования.

Эффективность сжатия существенно определяется тем, что таблицы в Sybase IQ сохраняются по колонкам, а не построчно.

В [4] указывается использование сжатия при представлении битовых индексов в Sybase IQ. В [25] отмечается, что битовые индексы сжимаются с помощью

[16] J. Ziv and A. Lempel. On the complexity of finite sequences. *IEEE Transactions on Information Theory,* 22(1):75-81, 1976.

того же словарного метода, что и данные. Из руководства администратору [57] следует, что система обеспечивает сжатие индексных структур любого вида с помощью метода сжатия одного типа.

Teradata

Комплекс Teradata фирмы NCR является реляционной СУБД и позиционируется на рынке производителем как средство ведения больших хранилищ данных. Таким образом, данная СУБД ориентирована на использование в системах поддержки принятия решений.

Teradata версии 2 Release 5.0 поддерживает словарное сжатие таблиц на уровне значений колонок [40]. Каждая колонка может быть независимо закодирована с помощью словаря наиболее часто используемых значений этого атрибута. Размер словаря должен быть равен 2^n-1, n = 1…8, и составляет максимум 255. Если атрибут может принимать неопределенное значение (NULL), то оно сжимается независимо. Словарь для каждой колонки сохраняется в заголовке таблицы. В заголовке каждой строки имеются флаги для указания факта сжатия значений колонок. Ссылка на фразу словаря для сжатого значения атрибута также записывается в заголовке строки.

Недостатком реализации метода является невозможность применения сжатия к атрибутам типов «строка переменной длины» (VARCHAR), «время» (TIME) и некоторых других более специфичных.

На основании ряда экспериментов над несколькими реальными базами данных в [40] указывается, что применяемый метод сжатия должен обеспечивать среднюю степень сжатия в 2 и более раз для типовых хранилищ данных, при этом ожидаемое уменьшение времени выполнения запросов составляет несколько десятков процентов.

В [40] также отмечается, что в Teradata версии 2 Release 4.0 словарь для отдельной колонки мог состоять только из одного элемента.

Сравнение и выводы

Фирмы-производители стараются избежать корректного сравнения их продукции с конкурирующим ПО и препятствуют появлению полной информации о поведении СУБД на стандартных тестовых наборах. Поэтому не представляется в настоящее время возможным сопоставить комплексы по степени сжатия и влиянию компрессии на скорость работы. Таблица 3 содержит сравнение систем по наличию сжатия данных и применяемым алгоритмам экономного кодирования.

Таблица 3

Система	Сжатие таблиц	Сжатие индексов	Уровень сжатия таблиц	Метод сжатия таблиц	Заявляемая степень сжатия
ADABAS, версия 7.1.2	X	X	Значение атрибута	Поля переменной длины, кодирование отсутствующих значений	2
DB2, версия 5	X	.	Строка	Кодирование по статическому словарю LZMW	2-2.3
Ingres, версия 2.6	X	X	Значение атрибута	Поля переменной длины, кодирование отсутствующих значений	?
Microsoft Access, ядро MS Jet 3.0	.	X	.	.	.
MySQL, версия 4.0.18	X	X	Значение атрибута	Кодирование по Хаффману, устранение серий нулей и пробелов, упаковка битов, словарь значений атрибута	1.7-3.3
Oracle, версия 9*i* Release 2	X	X	Значение атрибута	Статический словарь значений атрибута	1.4-3.1
SAS System, версия 8.2	X	.	Строка	Кодирование длин серий и словарное сжатие со скользящим окном	?
Sybase IQ, версия 11.2	X	X	Страница	Словарное сжатие со скользящим окном типа LZH	?
Teradata, версия 2 Release 5.0	X	?	Значение атрибута	Статический словарь значений атрибута	2

Примечание: “X” обозначает факт наличия сжатия, “.” — отсутствие сжатия, “?” — отсутствие достаточной информации.

Все рассмотренные промышленные системы управления данными поддерживают такой способ экономного кодирования информации, как использование полей переменной длины. Но лишь ряд СУБД реализует более сложные методы сжатия. Показательно, что в этом случае для сжатия табличных данных используются исключительно словарные методы. Предпочтение отдается сжатию на уровне значения атрибута, что упрощает работы по выборке и по кодированию-декодированию данных при выполнении запросов. С другой стороны, сжатие на уровне строк и страниц должно способствовать улучшению степени сжатия в первую очередь текстовых данных и может иметь в этом свое преимущество. Заявляемая производителями степень сжатия БД составляет 2 и более раз.

Таким образом, реализации методов сжатия в СУБД отстают от современных достижений в этой сфере научной деятельности. Поддержка сжатия часто связана с

ограничениями по типу данных и уменьшением функциональности сжатых объектов. Слабо используются прогрессивные разработки в сжатии числовых и структурированных текстовых данных, в том числе позволяющие оперировать непосредственно закодированными данными при обработке запроса. Не уделяется внимание вопросу сжатия результирующих наборов, позволяющего значительно повысить производительность в распределенных информационных системах с малой пропускной способностью каналов связи. Возможно, в существующих системах недостаточно эффективно решается задача построения плана выполнения запроса с учетом факта сжатия данных, поскольку в доступных источниках этот вопрос не акцентируется.

Заключение: перспективные направления в области использования сжатия данных в СУБД

Применение сжатия данных в СУБД является перспективной и развивающейся областью знаний. Практическая польза от использования сжатия, выражающаяся в сокращении расходов памяти и повышении производительности системы, перевешивает трудности, связанные с эффективной реализацией поддержки экономного кодирования. Это в итоге должно сломить консерватизм фирм-производителей СУБД и заставить реализовать полноценную поддержку сжатия данных в их продуктах. Скорее всего, этот процесс уже начался вследствие введения поддержки сжатия табличных данных в последней версии СУБД Oracle, являющейся основным продуктом соответствующего сектора рынка.

Вместе с тем имеется целый спектр актуальных проблем, определяющих перспективные направления исследований.

1. Слабо исследованы возможности создания эффективных методов сжатия, позволяющих учитывать структуру данных. Например, если БД содержит временные ряды некоторых показателей, что характерно для хранилищ данных, то существуют сильные зависимости как по вертикали (по столбцу), так и по горизонтали (по строке).
2. Открыт вопрос экономного кодирования результатов выполнения запросов. Задача актуальна в первую очередь для распределенных вычислительных систем, имеющих малую или недостаточную пропускную способность каналов связи.
3. Обеспечение действительно высокой производительности СУБД при использовании сжатия возможно только при комплексном изменении механизма выполнения запросов. В частности, оптимизация плана выполнения запроса должна производиться с максимально точным учетом всех эффектов использования сжатия. С другой стороны, используемый метод сжатия должен иметь способы адаптации, учитывающие статистику работы оптимизатора.
4. Увеличение разницы в скорости процессора и элементов памяти диктует необходимость пересмотра устоявшихся взглядов. Использование более сложных методов, требующих значительного объема вычислений, но предоставляющих бóльшую степень сжатия может повысить реальную производительность СУБД. По этой причине может быть целесообразен

переход от применения сжатия на уровне значения атрибута к уровню страницы.

Благодарности

Автор благодарен Вадиму Юкину и Роману Волынцу за дискуссию и ряд дельных замечаний и предложений. Также большое спасибо д-ру Ви-Кеонгу Нг (Wee-Keong Ng) и Жимону Грабовски (Szymon Grabowski) за предоставленные статьи.

Если вы обнаружили ошибки
или неточности в тексте,
напишите об этом, пожалуйста,
по адресу
ms@compression.ru
Максиму Смирнову

Предметный указатель

Литература

1. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. – М.: ДИАЛОГ-МИФИ, 2002. – 384 с.
2. Дейт К. Д. Введение в системы баз данных. – 7-е изд. – М.: Вильямс, 2001. – 1072 с.
3. Alsberg P. A. Space and Time Savings Through Large Data Base Compression and Dynamic Restructuring. *Proc. IEEE* 63(8):1114-1122, August 1975.
4. Amer-Yahia S. and Johnson T. Optimizing queries on compressed bitmaps. In *Proc. of VLDB*, pp. 329–338, 2000.
5. Antoshenkov G. Byte Aligned Data Compression. U.S. Patent No: 5,363,098, October 1993.
6. Antoshenkov G. Dictionary-based order-preserving string compression. *In VLDB Journal*, 6(1):26-39, 1997.
7. Babu S., Garofalakis M., Rastogi R. SPARTAN: A Model-Based Semantic Compression System for Massive Data Tables. *Proc. of ACM SIGMOD'2001*, Santa Barbara, CA, May 2001, pp. 283-294.
8. Bassiouni M., Mukherjee A., Tzannes N. Experiments on Improving the Compression of Special Data Types. *Proc. IEEE Data Compression Conference*, Snowbird, Utah, April 8-11, 1991, p. 433.
9. Bell T.C., Moffat A., and Witten I.H. Compressing the Digital Library. 1994.
10. Bruni, P. and Naidoo, R. DB2 for OS/390 and Data Compression. First Edition. *IBM Redbook, SG24-5261-00*, 1998, 170 p.
11. Buchsbaum A. L., Caldwell D. F., Church K. W., Fowler G. S., and Muthukrishnan S. Engineering the compression of massive tables: an experimental approach. *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 175-184, 2000.
12. Cannane A., Williams H. E., and Zobel J. A General-Purpose Compression Scheme for Databases. *Proc. IEEE Data Compression Conference*, p. 519, 1999.

13. Chan C.Y. and Ioannidis Y.E. An Efficient Bitmap Encoding Scheme for Selection Queries. *Proc. ACM SIGMOD Intl' Conference*, Philadelphia, Pennsylvania, June 1999, pp. 215-226.
14. Chen Z. Building Compressed Database Systems. A Dissertation Presented to the Faculty of the Graduate School of Cornell University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy, Aug. 2002.
15. Chen Z., Gehrke J., Korn F. Query Optimization in Compressed Database Systems. *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data*, pp. 271-282, Santa Barbara, CA, May 2001.
16. Chen Z., Seshadri P. An Algebraic Compression Framework for Query Results. *Proceedings of the International Conference on Data Engineering ICDE'99*, San Diego, CA, March, pp. 177-188, 1999.
17. Cockshott W. P., Gilchrist J., McGregor D., Murray P., and Wilson J. Compressed, Memory Resident, Databases. 1996.
18. Collins K. Jet 3.0 Performance Overview White Paper. *Microsoft Corporation. Jet Program Management*. 1995.
19. Computer Associates. Advantage Ingres Database Administrator's Guide 2.6. 2002.
20. Cormack G. V. Data Compression in Database Systems. *Comm. of ACM*, 28(12):1336-1342, December 1985.
21. Eggers S., Olken F., and Shoshani A. A Compression Technique for Large Statistical databases. *Proc. VLDB Conf.*, September 1981.
22. Eggers S., and Shoshani A. Efficient Access of Compressed Data Performance. *Proc. VLDB*, Montreal, Oct. 1980, p. 205.
23. Gallager R.G. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668-674, Nov. 1978.
24. Goldstein J. Improved query processing and data representation techniques. *A dissertation submitted in partial fulfillment of the requirements for the degree of doctor of philosophy (computer sciences) at the University of Wisconsin – Madison*. 1999.
25. Goldstein J., Ramakrishnan R., and Shaft U.. Compressing relations and indexes. *Proc. IEEE Conf. on Data Engineering*, Orlando, FL, USA, pp. 370-379, 1998.
26. Goyal K., Ramamritham K., Datta A., Thomas H. Indexing and Compression in Data Warehouses. *Technical Report*, *Indian Institute of Technology, Bombay*, April 1999.
27. Goyal K. B., Ramamritham K., Datta A., Thomas H. M. Indexing and Compression in Data Warehouses. *Proceedings of the Int'l. Workshop Design and Management of Data Warehouses '99*, Heidelberg, Germany, June 14-15, 1999.
28. Graefe G. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys* 25(2), June 1993, p. 73-170.
29. Graefe G. and Shapiro L. D. Data Compression and Database Performance. *In Proceedings of the ACM/IEEE-Computer Science Symposium on Applied Computing*, Kansas City, MO, 1991.
30. Hann R.. Ingres Frequently Asked Questions. 1997.
31. Hubley M., Fairchild A. Software AG Adabas. *Gartner Product Report*. 2001.

32. Iyer B. R. and Wilhite D. Data Compression Support in Databases. *In Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, pp. 695-704. 1994.
33. Johnson T. Performance Measurements of Compressed Bitmap Indices. *Proceedings of 25th International Conference on Very Large Data Bases*, September 7-10, 1999 (VLDB'99), Edinburgh, Scotland, UK, pp. 278-289.
34. Korn F., Jagadish H. V., and Faloutsos C. Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences. *1997 ACM SIGMOD International Conference on Management of Data*, pp. 289-300, Tucson, AZ, May 1997.
35. Lekatsas H. and Wolf W. Random Access Decompression using Binary Arithmetic Coding. *IEEE Data Compression Conference*, pp. 306-315, Snowbird, UT, March, 1999.
36. Li J., Rotem D., Wong H. A New Compression Method with Fast Searching on Large Databases. *Proceedings of 13th International Conference on Very Large Data Bases*, 1987, Brighton, pp. 311-318.
37. Miled Z. B., Li H., Bukhres O., Bem M., Jones R., Oppelt R.. Data Compression in a Pharmaceutical Drug Candidate Database. *Informatica 17*, 2001.
38. Moffat A. and Zobel J. Coding for compression in full-text retrieval systems. *In Proc. IEEE Data Compression Conference*, pp. 72-81, Snowbird, Utah, March 1992. IEEE Computer Society Press, Los Alamitos, California.
39. Moffat A. and Zobel J. Compression and fast indexing for multi-gigabyte text databases. *Australian Computer Journal*, 26(1):1-9, 1994.
40. Morris M. Teradata Multi-Value Compression V2R5.0. *A Teradata White Paper*, EB-3099, July 2002.
41. MySQL AB (2004). MySQL Reference Manual for version 4.0.18.
42. Ng W. K. and Ravishankar C. V. Block-Oriented Compression Techniques for Large Statistical Databases. *Knowledge and Data Engineering*, 9(2):314-328, 1997.
43. Ng W. K., Ravishankar C.V. Relational Database Compression Using Augmented Vector Quantization. *ICDE* 1995: 540-549.
44. Ng W. K., Ravishankar C.V., Chinya V. Data compression system and method representing records as differences between sorted domain ordinals representing field values. *US Patent 5,603,022*. Feb. 1997.
45. Oracle Corp. Oracle Online Documentation Library. 2001.
46. Poess M. Table Compression in Oracle9*i* Release 2: A Performance Analysis. *An Oracle White Paper*, January 2003.
47. Qiang J. Querying and Mining Semantic Compressed Databases. *A term paper submitted for PhD qualifier examination. School of Computing, National University of Singapore*, 2002.
48. Ray G. Data Compression in Databases. *Master's Thesis*, Dept. of Computer Science and Automation, Indian Institute of Science, June 1995.
49. Ray G., Haritsa J., and Seshadri S. Database compression: A performance enhancement tool. *In Proc. COMAD*, Pune, India, December 1995.
50. Roth M. A. and Van Horn S. Database compression. *ACM SIGMOD Record*, 22(3):31-39, Sept. 1993.

51. Ruth S. and Keutzer P. Database Compression for Large Business Files. *Datamation*, 18, Sept. 1972, p. 62.
52. SAS Institute Inc., SAS OnlineDoc®, Version 8, Cary, NC: SAS Institute Inc., 1999.
53. Software AG. ADABAS DBA Reference Manual. 1997.
54. Srivastava J., Ngo H. Statistical Databases. *Technical Report.*
55. Stockinger K. Multi-Dimensional Bitmap Indices for Optimising Data Access within Object Oriented Databases at CERN. *PhD*. Nov. 2001.
56. Stockinger K., Wu K., Shoshani A. Strategies for Processing ad hoc Queries on Large Data Warehouses. *Proc. DOLAP'02*, November 4–9, 2002, McLean, Virginia, USA.
57. Sybase, Inc. Sybase IQ Administration Guide, 1997.
58. Vo B.D., Vo K-P. Using Column Dependency to Compress Tables. *Proc. IEEE Data Compression Conference*, Snowbird, Utah, March 23-25, 2004, pp. 92-101.
59. Westmann T., Kossmann D., Helmer S., and Moerkotte G. The implementation and performance of compressed databases. *Technical Report 3/98*, Universitat Mannheim, 1998. Отчет был напечатан: Westmann T., Kossmann D., Helmer S., and Moerkotte G.. The Implementation and Performance of Compressed Databases. *SIGMOD Record*, 29(3):55-67, 2000.
60. Witten I.H., Bell T.C., and Nevill C.G.. Indexing and compressing full-text databases for CD-ROM. *Journal of Information Science*, 17:265-271, 1992.
61. Wu K., Otoo E., and Shoshani A. Compressing Bitmap Indexes for Faster Search Operations. *In Proceedings of SSDBM 2002 Preprint as LBNL-49627*, 2002.
62. Wu K., Otoo E., and Shoshani A. Compressed bitmap indices for efficient query processing. *Technical report LBNL/PUB-3161, Lawrence Berkeley National Laboratory, Berkeley, CA,* 2001.
63. Zandi A., Iyer B., and Langdon G. Sort Order Preserving Data Compression for Extended Alphabets. *IEEE Data Compression Conf.*, Snowbird, Utah, March 30 – April 1, 1993, pp. 330-339.
64. Ziv J., Lempel A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, Vol. 23(3), pp.337-343, May 1977.
65. Ziv J. and Lempel A. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, Vol. 24(5), pp.530-536, Sept. 1978.