ЧАСТЬ VI ЗЕЛЕНЫЕ БАЗЫ ДАННЫХ И ОБЛАЧНЫЙ КОМПЬЮТИНГ

РАЗДЕЛ 22 ОПТИМИЗАЦИЯ БАЗ ДАННЫХ ДЛЯ ЗЕЛЕНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

СОДЕРЖАНИЕ РАЗДЕЛА

22.1 Задачи оптимизации операций и структур в базах данных в контексте зеленых ИТ

22.1.1 Задачи оптимизации операций в базах данных

Известно, что основной формой организации информации в современных информационных системах (ИС) являются связки «база данных (БД) - система управления базой данных (СУБД)».

Парадигма БД сформировалась в середине 60-х годов XX столетия в результате многочисленных попыток избавиться от недостатков файловых систем организации информации, возникающих при многопользовательском доступе к данным. Основными из этих недостатков являются [1]:

1) изолированность данных (параллельно работающие приложения не могут одновременно изменять записи в одном и том же файле, а совместная обработка нескольких файлов достаточно сложная);

2)зависимость программ от данных (при изменении структуры файла требуется внесение соответствующих изменений в прикладную программу и последующая ее перекомпиляция);

3)дублирование данных (различные приложения, использующие данные об одном и том же объекте, хранимые в своих независимых файлах, непроизводительно расходуют память и могут привести к противоречивости данных);

4)отсутствие описания данных (в файлах данные, обрабатываемые прикладными программами, хранятся без их описания, что приводит к сложности документирования ИС и появлению ошибок).

Естественным средством устранения основных этих недостатков является отделение хранения данных от их обработки, что и реализуется связкой «БД-СУБД». В этой связке БД - это информационная модель предметной области, представленная в виде совокупности данных, хранимых в памяти компьютера и связанных между собой правилами, определяющими общие принципы их

описания, хранения и манипулирования, а СУБД - это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Отметим, что ядром БД является модель данных, т.е. совокупность структур данных (именно они обеспечивают возможность использования того или иного алгоритма) и операций их обработки.

Кроме того, СУБД по своему предназначению делятся на операционные (т.е. обладающие высокой скоростью реакции на запрос, извлечения и представления информации) и предназначенные для работы с хранилищами данных, содержащими очень большой объем информации, подготовка представления которой занимает значительное время.

Проблема проектирования связок «БД-СУБД» является одной из центральных в процессе разработки ИС, так как от ее решения, во многом, зависит качество создаваемой ИС. Этой проблеме посвящены усилия многочисленных исследователей во всем мире. Большое число подходов к ее решению (см., например, [2-4]) обусловлено следующим.

В результате стремительного развития средств вычислительной техники ИС стали применяться практически во всех сферах жизнедеятельности современного общества. Как следствие, происходит рост как многообразия моделей данных и способов их представления на физических носителях (т.е. типов БД), так и многообразия средств взаимодействия с БД (т.е. СУБД).

Основными требованиями, предъявляемыми к связкам «БД-СУБД» являются их безопасность (как ИС) и эффективность функционирования на протяжении всего жизненного цикла.

Проблеме обеспечения безопасности связок «БД-СУБД» посвящены многочисленные исследования, и в настоящее время в ее решении достигнуты значительные успехи [5].

Совершенно иная ситуация имеет место с проблемой обеспечения эффективности функционирования связок «БД-СУБД». На практике, как правило, разработчики руководствуются принципом «мы делаем так, как

умеем», не проводя детального теоретического анализа этой проблемы в конкретной ситуации. Такой подход обусловлен, прежде всего, большой внутренней сложностью этой проблемы, природа которой состоит в следующем. В терминах классической связки В.М. Глушкова «управляющий автомат - операционный автомат» [6] связка «БД-СУБД» представляет собой операционный автомат с переменной структурой. Как известно, анализ и синтез таких автоматов является одной из наименее изученных проблем теории автоматов. Кроме того, широкому спектру задач, решаемых с использованием информационных технологий, соответствует широкий спектр типов автоматов, являющихся моделями связок «БД-СУБД».

В настоящее время эффективность функционирования ИС, в частности, связок «БД-СУБД», включает минимизацию потребляемых ресурсов, понимаемых в широком смысле (причем как на этапе эксплуатации, так и на этапе разработки). Поэтому естественно, что проблема оптимизации операций связок «БД-СУБД» не только не потеряла свою актуальность, а получила «второе дыхание».

Отметим один принципиальный факт. Как было отмечено выше, удовлетворительное решение проблемы оптимизации операций в СУБД вносит свой вклад в общую проблематику Green IT. С другой стороны, применение накопленных моделей и методов оптимизации операций можно (и нужно) применять для решения задач собственно Green ГГ: вместо оценок времени работы, памяти и т.д. надо рассматривать в качестве мер сложности, например, потребляемую энергию. Другими словами, стандартные временные и емкостные сигнализирующие надо заменить на сигнализирующие, диктуемые спецификой Green IT.

22.1.2 Задачи оптимизации структур хранения данных

Концепция Green ГТ предполагает снижение энергозатрат при выполнении вычислительных задач. Для достижения этой цели необходимо стремиться к оптимальному использованию имеющего оборудования, такому как процессор, дисковые накопители и др., что достигается благодаря снижению

вычислительной сложности применяемых алгоритмов, оптимизации структур данных и другими комплексами мероприятий.

При хранении данных в реляционных базах данных для прикладных применений возможно использование различных моделей (схем) данных. Эти модели позволяют решать одну и ту же задачу различными, с точки зрения физической организации, способами, что позволяет выбирать оптимальные модели в зависимости от задачи, в которой они применяются.

Одной из типовых задач повышенной сложности, с которой сталкиваются архитекторы БД, является задача хранения иерархических данных. Реляционная модель изначально предназначена для связанных табличных данных, и не рассчитана на хранение иерархических данных. Применение для этого типа задач специализированных иерархических хранилищ не оправдано стоимостью их внедрения и слабой популярностью таких хранилищ. На практике эта задача решается путём использования иерархических структур в реляционных БД.

Существует несколько методов построения иерархических структур в реляционных БД. В ходе исследования были проанализированы четыре таких метода:

1. Вложенное множестве (Nested Set). Представляет собой дерево, которое отражает подчинение элементов одного уровня вложенности другому (упорядоченное по уровню вложенности).

2. Список смежных вершин (Adjacency List). Для реализации данной методики применяется хранение информации о связях «наследник- родитель» в виде специальной структуры данных со списком смежных вершин и уровнями этих вершин.

3. «Замкнутые» таблицы (Closure Tables). При этом подходе хранится полный перечень путей в дереве. Возмбжно хранение пути нулевой длины для связи узла с самим собой.

4. Материализованный путь (Materialized Path). Идея данного подхода заключается в хранении информации о полном пути от корня дерева к данному узлу в заранее подготовленном виде.

Исследование перечисленных методов хранения иерархических данных позволит выбрать метод, наилучший с точки зрения концепции зеленых ИТ.

Далее в разделе анализируются результаты оценивания эффективности функционирования связок «БД-СУБД» и выбора методов хранения иерархических данных.

22.2 Оптимизация операций в базах данных

22.2.1 Особенности процесса проектирования связки «БД-СУБД»

Процесс проектирования связки «БД-СУБД» состоит из следующих 4-х этапов:

1.Системный анализ предметной области.

2.Построение инфологической (*концептуальной*) *модели предметной области.* Эта модель не зависит от СУБД (и, следовательно, от среды хранения данных), и представляется в терминах выбранной семантической модели (одной из наиболее распространенных семантических моделей является разработанная П. Ченом ER-модель (Entity Relationship) [7,8]).

3. Построение логической модели. Выбирается модель данных и тип СУБД. Строится схема БД и подсхемы для различных пользователей. Создается набор возможных типовых запросов. Определяются спецификации для программного обеспечения (ПО).

4.Построение физической модели. Выбирается размещение БД на внешних носителях и определяется используемая СУБД. Создается реальная БД. Программируются и отлаживаются приложения, которые будут работать с БД.

Результатом процесса проектирования связки «БД-СУБД» является готовая к заполнению реальная БД и готовое к использованию ПО.

Таким образом, в связке «БД-СУБД» выделяются инфологический, логический и физический уровни. Первые два уровня называются внешними и предназначены для поддержки санкционированного доступа к данным БД.

Третий уровень называется внутренним и отображает организацию данных в среде хранения.

С позиции логической модели многообразие типов связок «БД- СУБД» в значительной мере определяется используемыми моделями данных. В настоящее время основными являются следующие модели данных.

1. *Иерархическая модель.* Разработана в [9]. Модель представляет собой набор ориентированных корневых деревьев. Каждая вершина дерева соответствует записи (или упорядоченному набору записей) со встроенными указателями на непосредственного предка (кроме корня дерева) и потомков вершины (кроме листьев). Поддерживает типы связей «один к одному» и «один ко многим». Модель эффективна при выполнении запросов, естественно формулируемых в терминах операций модификации двухсвязных списков.

2. *Сетевая модель.* Разработана в [10, 11]. Представляет собой набор ориентированных связных графов с рядом ограничений на их структуру. Расширение возможностей по сравнению с предыдущей моделью состоит в том, что за счет двух встроенных связей «один ко многим» поддерживается связь «многие ко многим». Модель эффективна при выполнении запросов, естественно формулируемых в терминах любых операций над двухсвязными списками.

3. *Реляционная модель.* Разработана в [12]. Модель основана на понятии «отношение» [13, 14] — одном из основных понятий математики. Представляет собой набор двумерных таблиц, строки которых имеют одинаковую структуру. Модель эффективна при выполнении запросов, естественно формулируемых в терминах операций над отношениями. В настоящее время эта модель де-факто является «промышленным стандартом».

4. *Объектно-ориентированная модель.* Возникла в результате формирования в 80-х годах XX столетия объектно-ориентированного подхода к программированию [15-17]. В этом подходе основными понятиями являются «объект» (имеющий уникальный идентификатор) и «литерал». Объекты (они разбиваются на атомарные и структурированные) инкапсулируют свое состояние (определяемое значениями набора свойств объекта) и поведение (т.е.

операции, которые могут быть выполнены объектом или над ним). Свойства объекта делятся на атрибуты (не являющиеся объектами и принимающими в качестве значения литерал или идентификатор) и связи (представленные посредством ссылочных атрибутов). Объекты, имеющие одинаковые атрибуты и отвечающие на одни и те же сообщения, образуют класс. Отношение наследования (в частности, множественное) дает возможность определять один класс как частный случай другого класса (соответственно других классов). Таким образом, логическая структура этой модели похожа на структуру иерархической модели (отличие состоит в методах манипулирования данными). В настоящее время объектно-ориентированная модель недостаточно проработана теоретически. Как следствие, отсутствует ее стандарт.

5. *Объектно-реляционная модель.* Представляет собой расширение реляционной модели за счет поддержки основных концепций (кроме наследования классов) объектно-ориентированного программирования [18]. В этой модели сняты ограничения неделимости (атомарности) значений атрибутов в таблицах, допускаются поля, значениями которых являются таблицы, встроенные в таблицы. Это дает возможность создавать типы данных любой степени сложности. Достоинством модели является возможность использовать существующие реляционные БД для вновь разрабатываемых приложений. В настоящее время на практике при объектно-ориентированном подходе к программированию используется именно объектно-реляционная модель.

6. *Многомерная модель.* Является обобщением реляционной модели. Представляет собой набор многомерных таблиц [19,20]. Модель предназначена для аналитической обработки больших объемов информации, связанной со временем (в частности, для решения задач прогнозирования). Основными понятиями являются «измерение» и «ячейка». Измерение состоит в выделении множества однотипных данных, соответствующих грани многомерной таблицы. Ячейка — это поле, значение которого определяется фиксированным набором измерений. Если все таблицы имеют одинаковую размерность и совпадающие типы данных при всех измерениях, то модель называется

гиперкубической, иначе — поликубической. В модели реализованы специальные операции «срез», «вращение», «агрегация» и «детализация», дающие возможность осуществлять анализ информации на различных уровнях ее обобщения.

С позиции физической модели многообразие типов связок «БД— СУБД» можно разбить на следующие классы.

1.По среде размещения. Связка «БД-СУБД» является:

1)локальной (сосредоточенной), если она реализована на одном компьютере;

2)распределенной, если она реализована в компьютерной сети.

2. По способу доступа. Связка «БД-СУБД» является:

1)мейнфреймовой, если рабочее место - это текстовый или графический терминал, а вся информация обрабатывается на сервере, т.е. компьютере или компьютерах, где хранится связка;

2)файл-серверной, если она размещена на сервере, доступ к данным и прикладным программам осуществляется через локальную сеть, а ядро СУБД реализовано на каждом клиентском компьютере;

3)клиент-серверной, если она размещена на сервере, доступ к данным осуществляется через локальную сеть, а клиентская часть СУБД и прикладные программы реализованы на клиентских компьютерах;

4)встраиваемой, если она является программной библиотекой, т.е. на локальных компьютерах организовано унифицированное хранение больших объемов данных, а доступ к ним происходит посредством запросов или вызовом функций библиотеки из приложения пользователя.

3.По подходу к организации вычислений. Связка «БД-СУБД» является:

1)классической (фон-неймановской), если при обработке запросов используется только классический (т.е. последовательный) подход к организации вычислений;

2)нео-классической, если при обработке запросов используются параллельные и/или распределенные вычисления.

В реальной связке «БД-СУБД» время обработки любого запроса состоит из (1) времени, затраченного на операцию загрузки данных с медленного носителя в оперативную память, (2) времени, затраченного на вычисление, и (3) времени, затраченного на возвращение результата в соответствующий медленный носитель. Поэтому естественно выделяются следующие два подхода к анализу эффективности обработки запросов в связке «БД-СУБД».

Первый подход основан на использовании симбиоза математической логики, теории моделей и прикладной теории алгоритмов. Предметом его исследования является математическая модель связки «БД-СУБД». Цель состоит в построении нижних оценок асимптотической временной сложности функционирования связки «в наихудшем случае» или «в среднем».

Второй подход основан исследовании реальной связки «БД-СУБД». Как правило, он осуществляется методами статистического анализа. Цель состоит в оценке реального времени обработки запросов (возможно, модельными) реализациями связки при тех или иных условиях.

Подчеркнем, что для получения адекватной картины оба эти подхода к анализу эффективности обработки запросов должны осуществляться как на этапе проектирования, так и в процессе эксплуатации связки «БД-СУБД», особенно при ее структурных модификациях и создании новых версий.

22.2.2 Введение в анализ сложности алгоритмов

В процессе работы любой алгоритм преобразования информации обрабатывает некоторые входные данные того или иного «размера», характеризующего «объем памяти», необходимой для их хранения. К определению «размера» применяется один из следующих двух подходов [21-23]. При первом подходе используется *равномерный вес,* т.е. предполагается, что для записи числа n используется единица памяти, а каждая элементарная операция выполняется в течение единицы времени.

При втором подходе используется *логарифмический вес*, т.е. предполагается, что для записи числа n используется $[\log n]$ единиц памяти, и

для каждой элементарной операция вычисляется время ее выполнения как функция от размера используемых в ней данных.

Ниже будет рассматриваться временная сложность алгоритмов, но, как отмечалось во введении подраздела, эту сигнализирующую можно заменять на другие сигнализирующие, отражающие специфику Green IT, на сам подход к оценке сложности алгоритмов это не повлияет.

Анализ временной сложности алгоритма при его работе над входом фиксированного размера при равномерном весе, состоит в оценке числа выполненных элементарных операций, а при логарифмическом весе - в оценке времени, затраченного на все сработавшие элементарные операции. Использование равномерного веса проще, чем использование логарифмического веса. При анализе временной сложности обработки запросов связкой «БД-СУБД» существенную роль играют операции ввода-вывода. Поэтому при таком анализе более адекватным является использование логарифмического веса. Отметим, что использование равномерного веса дает возможность получить некоторые грубые оценки временной сложности обработки запросов связкой «БД-СУБД». Однако, подчеркнем еще раз, в каждом случае вопрос о том, насколько приемлемы такие оценки, требует отдельного изучения.

Выделяют три типа временной сложности алгоритма при обработке данных фиксированного размера: *сложность в наихудшем случае, сложность для почти всех входов* и *сложность в среднем.* Исследование временной сложности алгоритма в наихудшем случае существенно проще, чем исследование его сложности для почти всех входов или сложности в среднем. В последнем случае предполагается, что для множества данных фиксированного размера задано адекватное распределение вероятностей появления его элементов при работе алгоритма (что является достаточно сложной задачей).

При анализе временной сложности обработки запросов связкой «БД-СУБД» наиболее приемлемым является использование сложности в среднем. Именно она дает возможность оценить эффективность обработки запросов связкой при условиях, определенных в техническом задании. Отметим, что

использование сложности в наихудшем случае или сложности для почти всех входов также дает возможность получить некоторые грубые оценки временной сложности обработки запросов связкой «БД-СУБД». Однако в каждом случае вопрос о том, насколько приемлемы такие оценки, требует отдельного изучения.

Вычисление временной сложности алгоритма (за исключением тривиальных случаев) приводит к громоздким, малообозримым формулам. Для того, чтобы избежать такой ситуации, оценивают порядок роста этой сложности при неограниченном росте размера входных данных. Такая оценка называется *асимптотической временной сложностью алгоритма.* При ее вычислении для оценки функции g(*n*) (n$\in$*N*, *N* - множество натуральных чисел) используют следующие асимптотические обозначения (где *f(n)*)- достаточно простая функция):

1) g(n)=o(f(n)) (n $\to \infty$) означает, что $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$;
2) g(n)= ω(*f(n)*) (*n*$\to \infty$) означает, что *f(n)*=o(g(n));
3) g(n)= O(f(n)) (n$\to \infty$) означает, что существуют такое число $n_0 \in N$ и константа c>0, что для всех n>n_0 истинно неравенство g(*n*)<c*f(n)*;
4) g(n)= Ω(*f(n)*) (n$\to \infty$) означает, что существуют такое число $n_0 c$ ***N*** и константа c>0, что для всех n>n_0 истинно неравенство g(*n*)>c*f*(*n*);
5) g(*n*)= Θ(*f(n)*) означает, что существуют такое число $n_0 \in$ **N** и константы $c_1 c_2$>0, что для всех n>n_0 истинны неравенства $c_1 f(n) <$ g(n) $< c_2 f(n)$.

При анализе асимптотической временной сложности алгоритма чаще всего предпринимается попытка получить оценку вида *O(f(n))* (ее легче получить, чем оценку вида Θ(*f*(*n*)), и она более информативная, чем оценки вида o*f*(*n*)), ω(*f*(*n*)) или Ω(*f*(*n*))).

Таким образом, для теоретического анализа эффективности обработки запросов в процессе эксплуатации связки «БД-СУБД» наиболее приемлемым является вычисление асимптотической временной сложности в среднем при использовании логарифмического веса.

Рассмотрим кратко структуры данных, используемые при анализе эффективности обработки запросов связкой «БД-СУБД» [23-25].

Массив представляет собой линейную структуру, состоящую из л «ячеек» (число л - размер массива), в каждую из которых может быть записан любой элемент, принадлежащий фиксированному множеству ***S*** однотипных элементов (ниже предполагается, что на множестве ***S*** задано равномерное распределение вероятностей). Выделяют следующие типы массивов:

1. Статический массив. Его размер ***n*** задаётся на стадии объявления и не может быть изменен во время работы алгоритма. Выборка или запись элемента по заданному индексу в такой массив осуществляется за время

$$T = \begin{cases} O(\log n), & \text{если } n \to \infty, |S| \text{фиксировано} \\ O(\log |S|, & \text{если } |S| \to \infty, n \text{ фиксировано}) \\ O(\log n - \log |S|), & \text{если } n \to \infty \text{ и } |S| \to \infty \end{cases} \qquad (22.1)$$

Вектор. Является динамическим массивом с возможностью изменения размера *n.* Он обеспечивает те же операции, что и статический массив, а также дает возможность удалять и добавлять элементы в концы вектора. Выделяют следующие типы векторов:

1)стек, если элемент может быть добавлен только в один конец вектора, и удален только из этого же конца вектора (отметим, что именно стек лежит в основе организации рекурсивных процедур);

2) очередь, если элемент может быть добавлен только в один конец вектора, а удален только из другого конца вектора;

3)дек, если добавление и удаление элемента возможно для любого из концов вектора.

Операция удаления элемента из вектора осуществляется за время, определенное формулой (22.1). С операцией добавления элемента в вектор ситуация следующая. Если размер *n* вектора меньше некоторого числа n_0 (определяемого используемым ПО), то время выполнения этой операции определяется формулой (22.1). Если же размер вектора достиг числа n_0, то для выполнения операции добавления элемента требуется выделить совершенно

новый кусок памяти и перенести туда все элементы вектора, т.е. операция добавления элемента выполнима за время

$$T = \begin{cases} O(n \log n), \text{если } n \to \infty, |S| \text{ фиксировано;} \\ O(n \log |S|, \text{если } |S| \to \infty, n \text{ фиксировано;}) \\ O(n \log n \cdot \log |S|, \text{если } n \to \infty \text{ и } |S| \to \infty.) \end{cases} \qquad (22.2)$$

3. Ассоциативный массив. Является динамической структурой, в которой реализована возможность обращения к значению по ключу (при этом ключом может быть практически любой тип данных). В современных системах web-разработки под массивом понимают именно ассоциативный массив.

4.Хеш-массив. Является ассоциативным массивом, в котором данные распределены по *m* «корзинам». Выбор корзины осуществляет хэш- функция, аргумент которой - ключ, а значение - номер корзины. Хэш- функция должна быть, во-первых, легко вычислимой, и, во-вторых, равномерно распределять ключи по корзинам. Такая конструкция дает возможность уменьшать время вычисления в определенное число раз.

Вектор является основой для построения *списков.* Выделяют следующие типы списков: -

1) *односвязный список* состоит из двух векторов одного и того же размера; 1-й вектор хранит элементы списка, а 2-й вектор — указатели либо на следующий элемент списка (правосторонний список), либо на предыдущий элемент списка (левосторонний список);

2) *двусвязный список* состоит из трех векторов одного и того же размера; 1-й вектор хранит элементы списка, 2-й вектор - указатели на предыдущий элемент списка, а 3-й вектор - указатели на следующий элемент списка.

Время вставки элемента в список - такое же, как и для вектора. Удаление элемента и расщепление списка на два подсписка по значению указателя, выполнимы за время $T=0(n \log n)$ $(n \to \infty)$. Конкатенация списков размеров n_1, и n_2 выполнима за время $T=0(n_1 \log n_1)$ $(n_1 \to \infty)$.

Списки представляют собой основу для построения эффективных структур данных, предназначенных для представления множеств, деревьев и графов (см., напр., [22-24]).

В современных связках «БД-СУБД» (особенно предназначенных для работы с хранилищами данных) используются специальные библиотеки функций, в основе которых лежит обработка списков.

Типичным таким примером является библиотека MapReduce [26-28], разработанная в компании Google. Ее основой являются функции шар и reduce. Входом функции тар является список значений и функция - трансформатор, а выходом — список результатов применения трансформатора к каждому значению. Вход функции reduce - список значений, стартовое значение аккумулятора и функция свертки (принимающая два значения-аргумента и возвращающая одно значение). Функция reduce последовательно передаёт в функцию свертки значение аккумулятора и очередное значение из списка, а результат помещает в аккумулятор. Результат выполнения функции reduce - финальное значение аккумулятора. Практика показывает, что широкий класс сложных задач обработки данных эффективно сводится к последовательному выполнению комбинаций функций тар и reduce.

22.2.3 Математические модели связок «БД-СУБД»

Формальный подход к теоретическому анализу алгоритмов преобразования информации основан на симбиозе математической логики, теории моделей и прикладной теории алгоритмов. Такой подход, по своей сути, состоит в следующем [29-31].

Над фиксированным множеством M конечных структур (т.е. моделей) сроится теория (по крайней мере 1-го порядка) T. Множество формул этой теории представляет собой некоторый язык L. Основными являются следующие три задачи:

1) *проверка выполнимости* (satisfiability checking), состоящая в том, чтобы ответить на вопрос: существует ли модель $S \in M$, для которой формула $\psi \in L$ выполнима?

2) *проверка модели* (model checking), состоящая в том, чтобы ответить на вопрос: верно ли, что для модели $S \in M$ и формулы $\psi \in L$ истинно соотношение $S \models \psi$ (т.е. формула ψ истинна в модели S)?

3) эволюция запроса (query evolution), состоящая в том, чтобы для модели $S \in M$ и формулы $\psi \vec{x} \in L$ (где $\vec{x}$ — множество свободных переменных) вычислить множество $\{\vec{a} \mid S \models \psi(\vec{a})\}$ (состоящее, говоря содержательно, из всех значений свободных переменных формулы ψ, при которых эта формула истинна в модели S).

Входными данными для проверки модели являются $S \in M$ и $\psi \in L$. Поэтому естественно возникают следующие задачи:

1) как выражается сложность решения задачи проверки модели через сложность модели $S \in M$ и формулы $\psi \in L$ (такая сложность называется комбинированной сложностью (combined complexity))?

2) какова сложность вычисления множества $\{\psi \in L \mid S, \models \psi\}$ для заданной модели $S \in M$ (такая сложность называется выразительной сложностью (expression complexity))?

3) какова сложность вычисления множества $(S \in M \mid S \models \psi\}$ для заданной формулы $\psi \in L$ (такая сложность называется структурной сложностью (structure complexity))?

В случае анализа связки «БД-СУБД» построение теории T осуществляется над некоторой алгеброй $A=(\Xi, B)$, где Ξ - множество структур данных, а B - множество базовых алгоритмов их обработки. Алгебра A представляет собой основу для построения некоторого множества (конечных) структур D. Объектом исследования является язык запросов L, а предметом исследования - анализ разрешимости и выразительности языка L [32-34], анализ временной сложности алгоритмов, представленных формулами языка L [35-37],

а также выделение тех классов формул языка *L,* которые представляют запросы, реализуемые с полиномиальной временной сложностью [38-40].

Такой подход исследовался в деталях для основных моделей данных: иерархической [41-43], сетевой [44-46] и реляционной [47-49]. В п. 22.1.1 было отмечено, что в настоящее время «промышленным стандартом» является реляционная модель связки «БД-СУБД». Этому обстоятельству (кроме успешных применений на практике) способствовало теоретическое обоснование преимущества реляционной модели перед остальными, установленное в процессе исследования перечисленных выше задач.

Кроме того, именно исследование указанных выше моделей данных сформировало Торию сложности интервального поиска в терминах информационных графов [50-52], предназначенную, в частности, для теоретического обоснования выбора физической организации данных.

Рассмотренный выше формальный подход получил дальнейшее развитие (и существенное усложнение) в процессе построения математических моделей *многомерной* связки «БД-СУБД».

В [20], исходя из анализа существующих моделей хранилищ данных [53-55], построена следующая общая модель многомерной связки «БД- СУБД» в терминах теории категорий, что дает возможность эффективно использовать язык схем.

Основа этой модели — многомерный объект. Он состоит из согласованных между собой схемы фактов S=(*F,D*) (где *F* — множество типов фактов, а *D* - частично упорядоченное множество категорий типов размерности, содержащее наименьший и наибольший элементы) и множеств *F,D* и *R*, соответственно, фактов, размерностей и отношений «факт-размерность». На многомерном объекте определена алгебра. Доказано, что эта алгебра замкнута (т.е. что результат выполнения операций над многомерным объектом — многомерный объект) и что она является не менее мощной, чем реляционная алгебра с функциями агрегирования.

Посредством операторов квантования времени транзакций и времени валидации выделяется согласованное во времени семейство многомерных

объектов. Для этого семейства развит математический аппарат, предназначенный для регулирования оценок запросов и представления неточных результатов. Показано, что выполнены все основные требования, предъявляемые к современным моделям многомерных связок «БД-СУБД». Рассмотрены особенности реализации построенной модели на основе использования технологии построения реляционных связок «БД-СУБД».

22.2.4 Алгоритмы оптимизации запросов

Одной из основных сложных задач, связанных с появлением связок «БД-СУБД», является «оптимизация обработки» запросов СУБД. Слово «оптимизация» является математическим термином и подразумевает выбор наилучшего (или близкого к нему в соответствии с заданным критерием) объекта или алгоритма, принадлежащего формально определенному классу, соответственно, объектов или алгоритмов. Ясно, что такой подход неприменим ни с теоретической, ни с практической точки зрения к связке «БД-СУБД» (как и ко многим другим прикладным системам). Поэтому под «оптимизацией обработки» запросов СУБД понимают стратегии, предназначенные для повышения эффективности процедур обработки запросов. Такие стратегии представляют собой эвристические методы, целесообразность использования которых обосновывается статистическими методами, т.е. обработкой результатов проведенных экспериментов.

Понятие «запрос», по своей сути, является выражением некоторого языка и рассматривается, по крайней мере, в следующих трех контекстах:

1) как стандартное требование доступа пользователя к данным БД (в этом случае «оптимизация» осуществляется за счет программирования соответствующих процедур поиска данных и преобразования входа пользователя в результат требуемого формата);

2) как транзакция, осуществляющая изменение данных БД на основании их текущего значения;

3) как выражение, которое СУБД использует для авторизации пользователя [56], обеспечения целостности данных [57] и синхронизации многопользовательского доступа к БД [58].

Для того, чтобы говорить об «оптимизации обработки» запросов, прежде всего требуется понимать, что понимается под «стоимостью» запроса. Выделяют следующие составляющие этого понятия:

1) коммуникационная стоимость, т.е. стоимость передачи данных из их местоположения во вторичную память (т.е. память, из которой загружаются данные для вычисления) плюс стоимость перемещения результата в его местоположение;

2)стоимость доступа к вторичной памяти, т.е. стоимость загрузки порций данных из вторичной памяти в основную память, используемую при вычислении;

3)стоимость запоминания, т.е. стоимость времени использования вторичной памяти и памяти буферов (что особенно актуально для связок «БД-СУБД», обладающих «узкими местами», изменяющимися в зависимости от запросов);

4)стоимость вычисления, т.е. стоимость времени, используемого непосредственно для вычисления.

Таким образом, «оптимизация обработки» запросов представляет собой нетривиальную многокритериальную задачу.

Методы «оптимизации обработки» запросов, разработанные в [59- 61], привели к пониманию необходимости разработки полномасштабного унифицированного подхода к решению этой задачи, основанного на нисходящем подходе (top-down approach). Анализ состояния этих исследований на 1984 г. представлен в обзоре [62].

В течение последнего 20-летия XX столетия сформировались следующие три направления исследования задачи «оптимизация обработки» запросов.

1. Разработка методов «оптимизации обработки» последовательности запросов. Формирование этого направления обусловлено попыткой снижения стоимости обработки запроса за счет использования того

фактора, что, как правило, в последовательности составных запросов содержится значительное количество общих подвыражений.

Первые методы «оптимизации обработки» последовательности запросов были основаны на исчерпывающемся поиске [63-65]. Эти методы использовали дважды экспоненциальную память от размера запроса и являлись неэффективными на практике. Ощутимый прорыв произошел в связи с разработкой методов «оптимизации обработки» последовательности запросов, основанных на использовании представления запросов ориентированными ациклическими И-ИЛИ графами (AND-OR DAG representation) [66-68]. Некоторые из них представляют собой прототипы методов, реализованных в SQL Microsoft Server. Более того, именно разработка этих методов дала возможность выработать ряд общих рекомендаций увеличения производительности SQL Microsoft Server 6.5, основные из которых состоят в следующем:

1) выделите серверу столько оперативной памяти, сколько он выдержит;

2) используйте массивы RAID (эти массивы распределяют запросы на чтение по нескольким физическим дискам) уровня 0 или 5 для распараллеливания получения информации из БД;

3) обеспечьте для функции Max Async I/O возможность использования всех преимуществ компьютера;

4) установите пороги расширения блокировок (Lock Escalation) на всю таблицу, позволяющие избежать «накладных расходов», связанных со значительным числом блокировок;

5) создайте кластеризованные индексы для запросов, считывающих диапазоны значений;

6) выделите некластеризованные индексы для запросов на поиск уникальных значений;

7) создайте составные индексы для поддержания последовательности запросов, что дает существенный выигрыш, когда, в основном, осуществляется чтение данных и выполняются операции UPDATE и INSERT;

8) индексируйте соединенные столбцы, что дает возможность существенно уменьшить (иногда на несколько порядков) время выполнения соединения таблиц;

9) используйте преимущества покрывающих индексов, т.е. содержащих все столбцы, указанные в операторах SELECT, UPDATE или DELETE, что уменьшает время выполнения этих операций до тех пор, пока суммарная длина всех входящих в индекс столбцов значительно меньше, чем длина строки таблицы.

2. *Разработка методов лексической и семантической «оптимизации» запросов.* Лексическая «оптимизация» запроса состоит в его преобразовании, направленном на устранение избыточности на основе анализа ограничений и условий, содержащихся в нем. Существуют следующие два подхода к решению этой задачи.

1) Преобразование запроса, представленного на нереляционном языке (а при необходимости и данных), к реляционной форме, и применение к ней методов лексической «оптимизации», разработанных для реляционных СУБД [69-71]. Обзор работ, посвященных таким методам, содержится в [72].

2) Построение новых алгебр, предназначенных для представления запросов с учетом особенностей новых языков, и разработка методов «оптимизации» выражений этих алгебр [73-75]. Обзор работ, посвященных таким методам, содержится в [76, 77].

Семантическая «оптимизация» запроса представляет собой валидацию и преобразование синтаксического дерева запроса к виду, пригодному для выполнения дальнейших шагов оптимизации. При этом осуществляется преобразование запросов в каноническую форму (т.е. раскрытие представлений, преобразование подзапросов в соединения, спуск предикатов), упрощение условий, распределение предикатов и преобразование дерева условий в пути выборки [78-80].

3. *Разработка методов «оптимизации обработки» запросов в параллельной распределенной связке «БД-СУБД».* По-видимому, именно эти

исследования определили следующие три направления исследований, наиболее интенсивно развивающиеся в настоящее время.

Во-первых, это разработка детерминированных и вероятностных методов «оптимизации» запросов для различных моделей параллельных распределенных связок «БД-СУБД» [81-83]. Отметим, что в работе [83] предложен способ выбора архитектуры параллельной распределенной связки «БД-СУБД» по критерию стоимости с ограничением на верхнюю границу доверительного интервала времени выполнения запроса. При этом для конфигурации SE [82, 84] исследована зависимость математического ожидания времени выполнения запроса от числа процессоров. Показано, что с ростом числа процессоров сначала время убывает благодаря распараллеливанию обработки запроса, а затем возрастает из-за перегрузки системы ввода/вывода, которая является разделяемым ресурсом.

Во-вторых, это построение моделей связок «БД-СУБД» на основе использования графических процессоров (GPU) [85-87]. В состав обычного CPU входят от 1 до 16 процессоров, а в состав GPU - несколько сотен простых процессоров. Под «простыми» понимаются процессоры, не содержащие компонент проверки условий, кэш и т.д. Архитектура GPU представляет собой SIMT (single instruction multiple thread). Имея функцию, выполняющую некоторую работу, разработчик назначает ее исполнение на тот или иной GPU. Модель программирования GPU описывает 6 типов областей памяти, различных по скорости доступа, прав на запись-чтение, объему. Наличие большого числа процессоров и быстрой памяти обеспечивает эффективность выборки данных со сложным условием, так как каждый поток исполняется независимо от других, оперируя только с малой частью обрабатываемых данных. В результате существенно снижается различие между временем загрузки данных с медленного носителя в оперативную память и временем, затрачиваемым на исполнения запроса и возвращения результата.

В-третьих, это разработка моделей и методов обработки и хранения больших данных. Под термином «большие данные» понимают электронные данные, характеризующиеся большим объемом, разнообразием и скоростью, с

которой структурированные и неструктурированные данные поступают по сетям передачи в процессоры и хранилища, а также наличием эффективных процессов переработки данных в требуемую информацию [88]. Типичными примерами больших данных являются системы компьютерной поддержки функционирования ведущих фондовых бирж мира, социальная сеть Facebook, проект Internet Archive, системы компьютерной поддержки уникальных экспериментальных установок исследования процессов микромира, таких, как Большой андронный коллайдер [89].

Именно актуальность разработки средств эффективной обработки больших данных привела к появлению концепции NoSQL (Not Only SQL) [90-92]. Ее суть состоит в проектировании архитектуры, обладающей свойством адаптации к возрастающим объемам данных.

При этом сохранение эффективности процессов обработки данных обеспечивается за счет высокой пропускной способности и потенциально неограниченного горизонтального масштабирования. В [93] предложена следующая классификация существующих в настоящее время NoSQL решений по типу используемых хранилищ.

1. *Хранилище ключ-значение.* Представляет собой «словарь», предназначенный для работы с данными по ключу. Дает возможность обеспечить высокую производительность обработки данных (за счет усложнения структуры запросов, так как в словаре отсутствует информация о структуре значений данных).

2 . *Документное хранилище.* Представляет собой словарь с логическим объединением множеств пар ключ-значение в структуру, называемую «документ». Документы могут иметь вложенную структуру и объединяться в коллекции. Работа с документами осуществляется по ключу и/или по значениям атрибутов.

3. Колоночные хранилища. Значения данных хранятся в виде интерпретируемых байтовых массивов, адресуемых кортежами вида (a_1, a_2, a_3), где a_1 - ключ строки, a_2 - ключ столбца и a_3 - метка времени [94]. Основой модели является «колонка». Число колонок в таблице потенциально

неограниченно. Колонки по ключам объединяются в семейства, обладающие заданным набором свойств.

Отметим, что значительное число публикаций, посвященных «оптимизации запросов» именно для этой модели (см., напр., [95-97]) обусловлено ее большой схожестью с реляционными связками «БД- СУБД».

4. Хранилища на графах. Используются для объектов с четко выраженной сетевой структурой (таких, как социальные сети, системы управления авиаперелетами, стратегические системы обеспечения безопасности страны, глобальные системы оповещения о природных катастрофах и т.д.). Модель состоит из вершин (в которых сосредоточены данные) и ребер (размеченных свойствами). Работа с данными осуществляется методом того или иного обхода графа по ребрам с заданными свойствами.

Обзор существующих технологий для работы с большими данными, содержится в [98]. Отметим, что в настоящее время уделяется значительное внимание разработкам, предназначенным для обработки потоков событий в реальном времени.