ФОРМАЛИЗАЦИЯ ИНФОРМАЦИИ И BIG DATA

Екатеринбург
2021

Министерство науки и высшего образования Российской Федерации
Уральский государственный экономический университет

ФОРМАЛИЗАЦИЯ ИНФОРМАЦИИ И BIG DATA

Рекомендовано
Советом по учебно-методическим вопросам и качеству образования
Уральского государственного экономического университета
в качестве учебного пособия

Екатеринбург
2021

УДК 681.3.06(075)
ББК 22.18я73
Ф79

Рецензенты:

департамент информационных технологий и автоматики
Института радиоэлектроники и информационных технологий — РТФ
Уральского федерального университета
имени первого Президента России Б. Н. Ельцина
(протокол № 2 от 18 февраля 2021 г.)

генеральный директор ООО «Октоника»
К. Г. Ведьманов

Авторский коллектив:
В. П. Часовских, *М. П. Воронов*, *В. Г. Лабунец*,
Е. Н. Стариков, *И. В. Иванов*

Ф79 **Формализация информации и big data** : учеб. пособие / авт. кол. : В. П. Часовских, М. П. Воронов, В. Г. Лабунец [и др.] ; М-во науки и высш. образования Рос. Федерации, Урал. гос. экон. ун-т. — Екатеринбург : Изд-во Урал. гос. экон. ун-та, 2021. — 218 с.

ISBN 978-5-9656-0312-1

Цель учебного пособия — формирование системы теоретических и практических знаний о современных информационных технологиях хранения и обработки больших данных (big data). Пособие включает общетеоретические аспекты формализации и использования различных моделей данных (от традиционных до новейших нереляционных систем), основных методов обработки больших наборов данных и функциональных возможностей СУБД Adabas для анализа больших наборов данных. Структура и содержание пособия полностью соответствуют требованиям федеральных государственных образовательных стандартов высшего образования (ФГОС ВО) и учебной программе дисциплины «Формализация информации и big data».

Для студентов очной и заочной форм обучения направления 02.03.03 «Математическое обеспечение и администрирование информационных систем» (бакалавриат). Также может быть полезно студентам направлений 09.03.03 «Прикладная информатика» (бакалавриат), 09.03.01 «Информатика и вычислительная техника» (бакалавриат), 38.04.05 «Бизнес-информатика» (магистратура), 09.04.03 «Прикладная информатика» (магистратура).

УДК 681.3.06(075)
ББК 22.18я73

ISBN 978-5-9656-0312-1

Введение

Мы живем в эпоху, которую часто называют веком информации. В наш век, поскольку мы верим, что информация ведет к эффективности и успеху, а также благодаря специфике новейших технологий (таких как Интернет вещей, веб-технологии, дополненная реальность и т. д.), мы собираем огромное количество информации.

Первоначально, с появлением компьютеров, мы начали собирать и хранить данные, рассчитывая на то, что компьютеры помогут разобраться в хаосе всевозможной информации. Но очень скоро массивные коллекции данных, хранящихся в разрозненных структурах, стали ошеломляющими по своим объемам. Этот первоначальный информационный хаос привел к созданию структурированных баз данных и систем управления базами данных (СУБД). Эффективные СУБД являются важным средством для управления большими массивами данных и особенно для эффективного и действенного извлечения конкретной информации из большой их коллекции. В свою очередь, распространение систем управления базами данных способствовало массовому сбору всевозможной информации.

Еще в 1950-х было сказано «ученые скоро утонут в информационном потопе», однако если сегодня сравнить объемы информации тогда и сейчас, тот «потоп» становится больше похож на лужу. Развитие информационных технологий и внедрение автоматизированных систем управления в организациях приводят к росту объемов хранимых и используемых данных. При создании современных систем принятия решений и в условиях постоянного повышения качества принимаемых решений организации строят все более совершенные математические и информационные модели, вовлекающие все больше исходных данных. Также интеллектуальные системы поддержки принятия

решений базируются на анализе решений, принятых в прошлом, и эффективности этих решений, таким образом, объемы хранимой информации еще более возрастают, поскольку возникает необходимость хранения статистических данных за большие периоды времени.

Сегодня мы имеем больше информации, чем можем обработать традиционными методами: от деловых операций и научных данных до спутниковых снимков, текстовых отчетов и данных военной разведки. Простого информационного поиска уже недостаточно для принятия решений. Столкнувшись с огромными массивами данных, мы создали новые потребности, которые помогут нам сделать лучший управленческий выбор. Эти потребности заключаются в автоматическом обобщении данных, извлечении «сущности» хранящейся информации и обнаружении закономерностей в необработанных данных.

Таким образом, остро актуальной является задача постоянного повышения эффективности использования информационных систем, обрабатывающих большие объемы данных. Предлагаемое учебное пособие посвящено данной проблематике. Оно включает общетеоретические аспекты формализации и использования различных моделей данных (от традиционных до новейших нереляционных систем), основных методов обработки больших наборов данных и функциональных возможностей СУБД Adabas для анализа больших наборов данных. Цель учебного пособия — формирование системы теоретических и практических знаний о современных информационных технологиях хранения и обработки больших данных (big data).

Проблема больших данных в последнее десятилетие активно обсуждается. Регулярно публикуются статьи, издано несколько десятков книг. Особенностью этого учебного пособия является системное представление моделей данных в разрезе их эволюции (от разбора понятий «данные» и «информация» — к проблематике больших данных, от первых традиционных моделей данных — к анализу новейших нереляционных систем). Проводится сравнительный анализ моделей данных наиболее популярных нереляционных СУБД и выявление предпочтительных сфер их применения. Существенную часть занимают во-

просы практической реализации методов хранения и обработки больших данных на примере промышленной СУБД Adabas.

Структура учебного пособия включает основной теоретический материал и обобщенные выводы по нему в конце каждой главы. С целью повышения эффективности освоения материала по каждой главе сформулированы вопросы для самоконтроля. В конце последней главы приведены небольшие практические задания, которые рекомендуется выполнить на основе примеров, разобранных в основном материале.

Пособие соответствует учебной программе дисциплины «Формализация информации и big data», изучаемой студентами направления 02.03.03 «Математическое обеспечение и администрирование информационных систем» (бакалавриат) очной и заочной форм обучения. Пособие также может использоваться студентами направлений 09.03.03 «Прикладная информатика» (бакалавриат), 09.03.01 «Информатика и вычислительная техника» (бакалавриат), 38.04.05 «Бизнес-информатика» (магистратура), 09.04.03 «Прикладная информатика» (магистратура) при изучении таких дисциплин, как «Базы данных», «Технологии обработки больших данных», «Разработка программных приложений».

Глава 1

Формализация больших данных — основные понятия и инструменты

1.1. Данные и информация

Изучение любой дисциплины начинается с формулировки определений ее фундаментальных терминов и категорий [1, с. 14].

Сообщения переносятся от источника к потребителю путем обмена веществом или энергией [2, с. 17]. В большинстве случаев переносчиком сообщения является энергия (электромагнитное колебание, электромагнитная волна), которая распространяется по каналу связи. В качестве переносчиков используются:

– гармонические (синусоидальные) колебания;

– колебания в виде последовательности прямоугольных импульсов.

Сам по себе переносчик не содержит информацию (сообщение). Любая информация, для того чтобы быть переданной или обработанной, должна быть соответствующим образом представлена (закодирована) в параметре переносчика (электромагнитного колебания) [2, с. 17].

Как только в переносчик закладывают информацию, электромагнитное колебание превращается в сигнал.

Сигнал — физический процесс энергетической природы, отображающий (несущий) сообщение. Параметр сигнала, изменение величины которого отображает передаваемое сообщение, называется информационным параметром. Примером может слу-

жить амплитуда электрического тока. Существуют две формы представления сигналов [2, с. 18]:

– аналоговая форма, при которой сигнал описывается непрерывной функцией, зависящей от времени;

– цифровая форма, при которой сигнал представляется совокупностью символов 0 и 1 (обозначения являются условными).

Параметры сигнала, зарегистрированные на материальном носителе, называются данными. С этой точки зрения под данными понимаются сведения, представленные в формализованном виде, позволяющем осуществить их хранение, передачу и обработку техническими средствами, например средствами вычислительной техники. Среди особенностей данных можно указать следующие [2, с. 18–19]:

– данные являются знаковой формой представления информации;

– данные могут рассматриваться как результат фиксации информации на некотором носителе;

– обработка данных с помощью компьютерных средств осуществляется без учета их смыслового содержания;

– данные не требуют интерпретации и осознания; осмысленные человеком данные представляют собой информацию;

– форма данных может быть различна в зависимости от выбранной знаковой системы и способа кодирования информации;

– данные характеризуются определенной структурой, определяемой выбранной знаковой системой и способом кодирования.

Данные, не являясь информацией как таковой, несут информацию о событиях. Только человек имеет возможность получить информацию на основе данных [2, с. 19].

Термин «информация» происходит от латинского слова *informatio*, изначально означающего разъяснение, осведомление, изложение. Особенностью термина «информация» является то, что, с одной стороны, он является интуитивно понятным практически для всех, а с другой — общепризнанной его трактовки в научной литературе не существует. Поэтому приведем несколько определений этого понятия, предложенных основоположниками кибернетики и информатики [1, с. 14–15].

Норберт Винер: «Информация — это обозначение содержания, черпаемого нами из внешнего мира в процессе нашего приспособления к нему и приведения в соответствие с ним нашего мышления».

Информация — это сообщение, неразрывно связанное с управлением, сигналы в единстве синтаксических, семантических и прагматических характеристик.

Информация — это передача, отражение разнообразия в любых объектах и процессах (неживой и живой природы).

Отдельные данные и сообщения обрабатывают, преобразовывают, систематизируют, сортируют и получают новую информацию, или новые знания.

Следует особо отметить, что как научная категория «информация» составляет предмет научения для самых различных областей знания: философии, информатики, теории систем, кибернетики и т. д.

Сама по себе информация может быть отнесена к абстрактным понятиям типа математических. Однако ряд ее особенностей приближает информацию к материальному миру. Так, информацию можно получить, записать, передать, стереть. Информация не может возникнуть из ничего. Но есть и особенности, отличающие информацию от реального мира. При передаче информации из одной системы в другую количество ее в передающей системе не уменьшается, хотя в принимающей системе оно, как правило, увеличивается. Кроме того, наблюдается независимость информации от ее носителя, так как возможны ее преобразование и передача по различным физическим средам с помощью разнообразных физических сигналов безотносительно к ее семантике, т. е. содержательности, смыслу. Информация о любом материальном объекте может быть получена путем наблюдения, натурного или вычислительного эксперимента, или путем логического вывода. В связи с этим информацию делят на доопытную, или априорную, и послеопытную, или апостериорную, полученную в результате проведенного эксперимента [1, с. 15−16].

Понятия «данные» и «знания» стоят в одном терминологическом ряду с информацией.

Под данными обычно понимают информацию, представленную в конкретных формах, которые адекватны возможным (ожидаемым) процессам ее обработки. Знания — это информация, на основании которой путем логических рассуждений могут быть получены определенные выводы [1, с. 16].

Также под знаниями иногда понимают данные, имеющие сложную организацию, обладающие как фактографической, так и семантической составляющими. Под фактографической составляющей понимается информация, связанная с регистрацией событий и явлений, а под семантической — информация, касающаяся содержательного (смыслового) толкования зарегистрированных фактов.

Получатель информации оценивает ее в зависимости от того, для какой задачи информация будет использована. Поэтому информация имеет свойство относительности. Одна и та же информация для одного получателя имеет глубокий смысл и обладает чрезвычайной ценностью, а для другого — является либо давно уже известной, либо бесполезной. Например, информация о последних достижениях в физике частиц высоких энергий очень важна для физика-ядерщика и совершенно бесполезна для агронома [1, с. 17].

При оценке информации различают такие ее аспекты, как синтаксический, семантический и прагматический [1, с. 17–18].

Синтаксический аспект связан со способом представления информации вне зависимости от ее смысловых и потребительских качеств. На синтаксическом уровне рассматриваются формы представления информации для ее передачи и хранения. Обычно информация, предназначенная для передачи, называется сообщением. Сообщение может быть представлено в виде знаков и символов, преобразовано в электрическую форму, закодировано, т. е. представлено в виде определенной последовательности электрических сигналов, однозначно отображающих передаваемое сообщение, и промодулировано для того, чтобы имелась возможность его передачи по выбранному каналу связи. Характеристики процессов преобразования сообщения для его передачи определяют синтаксический аспект информации при ее передаче. При хранении синтаксический аспект определяется другими формами представления информации, которые позволяют

наилучшим образом осуществить поиск, запись, обновление, изменение информации в информационной базе. Информацию, рассмотренную только относительно синтаксического аспекта, часто называют данными.

Семантический аспект передает смысловое содержание информации и соотносит ее с ранее имевшейся информацией. Смысловые связи между словами или другими элементами языка отражает тезаурус — словарь. Тезаурус состоит из двух частей: списка слов и устойчивых словосочетаний, сгруппированных по смыслу, и некоторого ключа, например алфавитного, позволяющего расположить слова в определенном порядке. При получении информации тезаурус может изменяться, и степень этого изменения характеризует воспринятое количество информации.

Прагматический аспект определяет возможность достижения поставленной цели с учетом полученной информации. Этот аспект отражает потребительские свойства информации. Если информация оказалась ценной, поведение ее потребителя меняется в нужном направлении. Проявляется прагматический аспект информации только при наличии единства информации (объекта), потребителя и поставленной цели.

1.2. Понятие больших данных

Для «больших данных» нет строгого определения. Изначально идея состояла в том, что объем информации настолько вырос, что рассматриваемое количество уже фактически не помещалось в памяти компьютера, используемой для обработки, поэтому инженерам потребовалось модернизировать инструменты для анализа всех данных. Так появились новые технологии обработки, например, модель MapReduce компании Google и ее аналог с открытым исходным кодом — Hadoop от компании Yahoo. Они дали возможность управлять намного бóльшим количеством данных, чем прежде. При этом их не нужно было выстраивать в аккуратные ряды или классические таблицы баз данных. На горизонте также появились другие технологии обработки данных, которые обходились без прежней жесткой иерар-

хии и однородности. В то же время интернет-компании, имеющие возможность собирать огромные массивы данных и острый финансовый стимул для их анализа, стали ведущими пользователями новейших технологий обработки, вытесняя компании, которые порой имели на десятки лет больше опыта, но работали автономно [3, с. 11].

По сути, большие данные представляют собой три шага к новому способу анализа информации, которые трансформируют наше представление об обществе и его организации [3, с. 16].

Первый шаг. В мире больших данных мы можем проанализировать огромное количество данных, а в некоторых случаях — обработать все данные, касающиеся того или иного явления, а не полагаться на случайные выборки. Начиная с XIX века, сталкиваясь с большими числами, общество полагалось на метод выборки. Сейчас он воспринимается как пережиток времен дефицита информации, продукт естественных ограничений для взаимодействия с информацией в «аналоговую эпоху». Понять искусственность этих ограничений, которые по большей части принимались как должное, удалось только после того, как высокопроизводительные цифровые технологии получили широкое распространение. Используя все данные, мы получаем более точный результат и можем увидеть нюансы, недоступные при ограничении небольшим объемом данных. Большие данные дают особенно четкое представление о деталях подкатегорий и сегментов, которые невозможно оценить с помощью выборки.

Второй шаг. Принимая во внимание гораздо бóльший объем данных, мы можем снизить свои претензии к точности. Когда возможность измерения ограничена, подсчитываются только самые важные показатели, и стремление получить точное число вполне целесообразно. Вряд ли вы сумеете продать скот покупателю, если он не уверен, сколько голов в стаде — 100 или только 80. До недавнего времени все наши цифровые инструменты были основаны на точности: мы считали, что системы баз данных должны извлекать записи, идеально соответствующие нашим запросам, равно как числа вносятся в столбцы электронных таблиц. Этот способ мышления свойствен среде «малых данных». Измерялось так мало показателей, что следовало как можно точнее подсчитывать все записанное. В некотором смысле мы уже

ощутили разницу: небольшой магазин в состоянии «подбить кассу» к концу дня вплоть до копейки, но мы не стали бы делать то же самое с валовым внутренним продуктом страны. Чем больше масштаб, тем меньше мы гонимся за точностью. Точность требует тщательной проверки. Она подходит для небольших объемов данных и в некоторых случаях, безусловно, необходима (например, чтобы проверить, достаточно ли средств на банковском счету, и выписать чек). Но в мире больших данных строгая точность невозможна, а порой и нежелательна. Если мы оперируем данными, большинство которых постоянно меняется, абсолютная точность уходит на второй план. Большие данные неупорядочены, далеко не все одинакового качества и разбросаны по бесчисленным серверам по всему миру. Имея дело с большими данными, как правило, приходится довольствоваться общим представлением, а не пониманием явления вплоть до дюйма, копейки или молекулы. Мы не отказываемся от точности как таковой, а лишь снижаем свою приверженность к ней. То, что мы теряем из-за неточности на микроуровне, позволяет нам делать открытия на макроуровне.

Третий шаг. Отход от вековых традиций поиска причинности. Люди привыкли во всем искать причины, даже если установить их не так просто или малополезно. С другой стороны, в мире больших данных мы больше не обязаны цепляться за причинность. Вместо этого мы можем находить корреляции между данными, которые открывают перед нами новые неоценимые знания. Корреляции не могут сказать нам точно, почему происходит то или иное событие, зато предупреждают о том, какого оно рода. И в большинстве случаев этого вполне достаточно.

Большие данные — важный шаг человечества в постоянном стремлении количественно измерить и постичь окружающий мир. То, что прежде невозможно было измерять, хранить, анализировать и распространять, находит свое выражение в виде данных. Использование огромных массивов данных вместо их малой доли и выбор количества в ущерб точности открывают путь к новым способам понимания мира. Это подталкивает общество к отказу от освященного веками поиска причинности и в большинстве случаев к использованию преимуществ корреляций [3, с. 19].

Буквально под термином «большие данные» понимают огромный объем хранящейся на каком-либо носителе информации. Причем данный объем настолько велик, что обрабатывать его с помощью привычных программных или аппаратных средств нецелесообразно, а в некоторых случаях и невозможно.

Большие данные — это не только сами данные, но и технологии их обработки и использования, методы поиска необходимой информации в больших массивах. Проблема больших данных по-прежнему остается открытой и жизненно важной для любых систем, десятилетиями накапливающих самую разнообразную информацию.

Тем не менее «большие данные» предполагают нечто большее, чем просто анализ значительных объемов информации. Проблема не в том, что организации создают огромные объемы данных, а в том, что бóльшая их часть представлена в формате, плохо соответствующем традиционному структурированному формату БД, — это веб-журналы, видеозаписи, текстовые документы, машинный код или, например, геопространственные данные. Все это хранится во множестве разнообразных хранилищ, иногда даже за пределами организации. В результате корпорации могут иметь доступ к огромному объему своих данных и не иметь необходимых инструментов, чтобы установить взаимосвязи между этими данными и сделать на их основе значимые выводы. Добавьте сюда то обстоятельство, что данные сейчас обновляются все чаще и чаще, и вы получите ситуацию, в которой традиционные методы анализа информации не могут угнаться за огромными объемами постоянно обновляемых данных, что в итоге и открывает дорогу технологиям больших данных [4].

Таким образом, «большие данные» включают:

– сами данные;
– технологии обработки данных;
– методы поиска.

1.3. Основные источники больших данных

В настоящее время накоплено огромное количество данных, от простых числовых измерений и текстовых документов

до более сложной информации, такой как пространственные данные, мультимедийные каналы и гипертекстовые документы. Вот частичный список разнообразной информации, собранной в цифровом виде в базах данных и файлах [5]:

– деловые операции. Каждая сделка в бизнес-индустрии (часто) фиксируется на долгое время. Такие операции обычно связаны со временем и могут быть межбизнесовыми сделками, такими как покупки, обмены, банковские операции, акции и т. д., или внутрифирменными операциями, такими как управление собственными товарами и активами. Крупные универмаги, например, благодаря широкому использованию штрих-кодов ежедневно производят миллионы транзакций, представляющих зачастую терабайты данных. Пространство для хранения данных не главное, но эффективное использование данных в разумные сроки для принятия конкурентных решений, безусловно, является самой важной проблемой для бизнеса, который борется за выживание в высококонкурентном мире;

– научные данные. В швейцарской лаборатории ядерных ускорителей при подсчете частиц, в канадском лесу при изучении показаний радиоприемника медведя гризли, на айсберге возле Южного полюса при сборе данных об океанической активности или в американском университете, изучающем человеческую психологию, наше общество накапливает колоссальные объемы научных данных, которые необходимо анализировать. К сожалению, мы получаем (и вынуждены хранить) новые данные быстрее, чем можем проанализировать старые, уже накопленные;

– медицинские и личные данные. От переписи населения до личных дел персонала и клиентов постоянно собираются очень большие объемы информации об отдельных лицах и группах. Правительства, компании и организации, такие как больницы, накапливают ценные объемы персональных данных, чтобы помочь им управлять человеческими ресурсами, лучше понимать рынок или удовлетворять нужды клиентов. Независимо от проблем конфиденциальности, которые часто выявляет этот тип данных, эта информация собирается, используется и даже передается. При корреляции с другими данными эта информация может пролить свет на поведение клиентов и т. п.;

– видеонаблюдение и фотографии. Существует тенденция хранить записи для дальнейшего использования и анализа;

– спутниковое зондирование. Существует бесчисленное количество спутников над всем земным шаром: одни из них геостационарны над областью, а другие вращаются вокруг Земли, но все они посылают непрерывный поток данных на поверхность. К примеру, организация NASA, которая контролирует большое количество спутников, получает ежесекундно больше данных, чем могут обработать все ее исследователи и инженеры. Многие спутниковые снимки и данные публикуются сразу же после их получения в надежде, что другие исследователи смогут их проанализировать;

– игры. Наше общество собирает огромное количество данных и статистики об играх, игроках и спортсменах. От хоккейных очков, баскетбольных пасов и провалов в автогонках до времени плавания, толчков боксера и шахматных позиций — все данные хранятся. Комментаторы и журналисты используют эту информацию для репортажей, но тренеры и спортсмены хотели бы использовать эти данные для повышения производительности и лучшего понимания оппонентов;

– цифровые медиа. Многие радиостанции, телевизионные каналы и киностудии оцифровывают свои аудиозаписи и видеофильмы, чтобы улучшить управление мультимедийными активами. Такие ассоциации, как НХЛ и НБА, уже имеют огромные коллекции игр в цифровом формате;

– САПР и программное обеспечение инженерных данных. Существует множество систем компьютерного проектирования (САПР) для архитекторов, проектирующих здания, или инженеров, разрабатывающих системные компоненты или схемы. Эти системы генерируют огромное количество данных. Кроме того, программная инженерия является источником значительных сходных данных с кодом, функциональными библиотеками, объектами и т. д., которые нуждаются в мощных инструментах для управления и обслуживания;

– виртуальные миры, дополненная реальность. Есть множество приложений, использующих трехмерные виртуальные пространства. Эти пространства и объекты, которые они содержат, описываются специальными языками, такими как VRML.

В идеале эти виртуальные пространства описываются таким образом, чтобы они могли совместно использовать объекты и места. Соответственно существует значительное количество доступных хранилищ объектов виртуальной реальности и пространства. Управление этими хранилищами, а также поиск и извлечение контента из них по-прежнему являются исследовательскими вопросами, в то время как размер коллекций продолжает расти;

– текстовые отчеты, записи, сообщения электронной почты. Большинство сообщений внутри и между компаниями или исследовательскими организациями или даже частными лицами основаны на отчетах и записках в текстовых формах, которыми часто обмениваются по электронной почте. Эти сообщения регулярно хранятся в цифровом виде для дальнейшего использования, например, в качестве справочных материалов, создавая огромные цифровые библиотеки;

– хранилища Всемирной паутины. С момента создания Всемирной паутины в 1993 г. документы всех видов форматов, содержания и описания были собраны и взаимосвязаны с гиперссылками, что делает ее крупнейшим хранилищем данных, когда-либо построенным. Несмотря на свой динамичный и неструктурированный характер, свою неоднородность и очень часто избыточность и непоследовательность, Всемирная паутина является наиболее важным источником данных, регулярно используемым для справочных целей, благодаря широкому разнообразию охватываемых тем и бесконечному вкладу ресурсов и издателей. Многие считают, что Всемирная паутина стала компиляцией человеческих знаний;

– значения данных радиочастотной идентификации (RFID);

– значения данных, генерируемых интеллектуальными сетями коммунальных предприятий;

– значения данных, полученных от датчиков промышленных двигателей и оборудования;

– данные, полученные из социальных сетей;

– многие другие.

1.4. Проблемы больших данных

Рассмотрев специфику и источники «больших данных», можно выявить и основные проблемы, связанные с их обработкой и анализом.

1. Получение информации об объеме и достоверности имеющихся данных. В информационных хранилищах предполагается, что информация извлекается из оперативных баз данных, преобразуется к необходимому виду, проверяется и только затем загружается в систему. Перечисленные операции выполняются с некоторой периодичностью, и здесь сразу же возникает вопрос: всегда ли возможна такая «периодичность» при работе с «большими данными», которые потоком поступают на вход и должны быть доступны для анализа как можно раньше? Может оказаться, что промежуток времени между появлением информации и ее доступностью для анализа меньше, чем время необходимое для выполнения операций по построению информационного хранилища [6]. Таким образом, мы не можем быть абсолютно уверенными в объеме и достоверности тех данных, которые мы анализируем, поэтому необходимы средства интеллектуального анализа с высоким уровнем быстродействия.

2. Разрозненность данных. Данные хранятся как отдельные, не связанные друг с другом файлы, поэтому требуется механизм для интеграции и систематизации больших объемов данных.

В информационных хранилищах неявно подразумевается, что при предварительной обработке данных (например, при поиске несоответствий) может использоваться ранее накопленное содержимое хранилища, что трудновыполнимо при работе с «большими данными». Проблема заключается прежде всего в том, что они всегда распределены, причем не так, как было бы удобно для анализа, а так, как было удобно их собирать. Например, если речь идет о телекоммуникационных системах, то данные «складываются» на региональных серверах. С точки зрения анализа удобнее распределять данные не по территориальному, а по временному признаку (каждый сервер отвечает за конкретный промежуток времени) и т. п. Однако информация станет до-

ступной для анализа только после того, как будет перенесена на необходимые серверы [6].

Особо следует отметить множественность форматов данных, которая сама по себе создает проблемы даже при не очень большом их объеме. Это мотивирует разработку специальных информационных конструкций и моделей информационных взаимодействий, которые часто отображают свойства информационного пространства или свойства поля [7].

3. Неточность данных (в отношении персональных данных). В настоящее время наиболее часто персональные данные вводятся пользователями самостоятельно. Как следствие, данные могут быть искажены (ввиду ошибок либо намеренно). Таким образом, должны присутствовать механизмы дополнительной проверки персональных данных.

4. Проблема конфиденциальности личных данных. Данные, собираемые триллионами сенсорных датчиков по всей планете, находятся вне нашего контроля.

С одной стороны, компании ищут более надежные способы защиты конфиденциальной информации и все чаще просят людей использовать биометрические данные в качестве уникального средства доступа к отдельным услугам или для обеспечения безопасности. С другой стороны, большая часть социальных данных создается в сотрудничестве между людьми в ходе совместных занятий по интересам или взаимоотношений. Контроль за сбором информации в целях защиты частной жизни человека потребовал бы установления режима своего рода полицейского государства, в котором были бы ограничены многие другие права на самовыражение. К тому же три основных варианта защиты от нежелательного использования сенсорных данных далеки от совершенства [8, с. 159].

Криптографические средства, разрешающие доступ к данным только обладателям электронных ключей, не годятся в большинстве случаев распространения информации, например, при размещении фото в Instagram. Социальные нормы, определяющие уместность распространения и использования информации, не защищают от злоумышленников. Остается защита, предусмотренная законодательными нормами [8, с. 159].

Соответственно, нужны инструменты, позволяющие установить, что дискриминация была обусловлена использованием социальных данных. Каждый запрос к базе данных является единицей информации, используемой инфопереработчиками в целях совершенствования своих продуктов и сервисов, и эти данные тоже могут быть задействованы для обеспечения нашей безопасности. Важность этой задачи будет расти по мере того, как алгоритмы начнут применяться не только для определения нашего физического местопребывания, но и для выводов о том, в каком состоянии духа мы находимся [8, с. 160].

5. Проблема потери информации. При создании приложений, работающих с большими данными, приходится сталкиваться со следующими характеристиками: большие объемы данных, интенсифицированные потоки данных, высокая структурная сложность, нелинейность моделей, требование существенного сокращения времени анализа или обработки данных и т. д. И если в прежнее время появление новых фактов легко фиксировалось и становилось предметом исследования, то в настоящее время проблемой становиться нахождение таких новых фактов и их формализация в больших массивах данных [7, с. 8].

6. Затраты на обработку «больших данных». Технологии требуют существенных капиталовложений. У половины компаний из числа использующих технологии обработки больших наборов данных затраты на них составляют от 20 до 30 %.

1.5. Способы хранения и представления данных

Как правило, интеллектуальный анализ данных не является специфичным для одного типа носителей или данных. Интеллектуальный анализ данных может быть применим к любому виду информационного хранилища. Однако алгоритмы и подходы могут различаться при их применении к разным типам данных. Действительно, проблемы, связанные с теми или иными типами данных, существенно различаются. Интеллектуальный анализ данных используется и изучается для реляционных, объектно-реляционных и объектно-ориентированных баз данных, хранилищ данных, транзакционных баз данных, неструктуриро-

ванных и полуструктурированных хранилищ, таких как Всемирная паутина, для прогрессивных баз данных — пространственных, мультимедийных, баз данных временных рядов и текстовых баз данных и даже для однородных файлов. Рассмотрим несколько примеров [5].

1. Однородные файлы. Являются наиболее распространенным источником данных для алгоритмов интеллектуального анализа данных, особенно на уровне исследований. Однородные файлы — это простые файлы данных в текстовом или двоичном формате со структурой, известной применяемому алгоритму интеллектуального анализа. Данные в этих файлах могут быть транзакциями, данными временных рядов, научными измерениями и т. д. [5].

2. Реляционные базы данных. Реляционная база данных состоит из набора таблиц, содержащих либо значения атрибутов сущностей, либо значения атрибутов из отношений сущностей. Таблицы имеют столбцы и строки, где столбцы представляют атрибуты, а строки — кортежи. Кортеж в реляционной таблице соответствует либо объекту, либо связи между объектами и идентифицируется набором значений атрибутов, представляющих уникальный ключ.

Наиболее часто используемым языком запросов для реляционных баз данных является SQL (Structured Query Language, язык структурированных запросов), который позволяет извлекать и манипулировать данными, хранящимися в таблицах, а также вычислять агрегатные функции, такие как average, sum, min, max и count. Например, SQL-запрос для выбора видео, сгруппированных по категориям, будет следующим:

SELECT count(*) FROM Items
WHERE type=video GROUP BY category.

Алгоритмы интеллектуального анализа данных, использующие реляционные базы данных, могут быть более универсальными, чем алгоритмы, специально написанные для однородных файлов, поскольку они могут использовать преимущества структуры, присущей этим базам данных. Хотя интеллектуальный анализ данных может извлечь выгоду из SQL для отбора, преобразования и консолидации данных, он выходит за рамки того, что

может предоставить SQL, например, прогнозирование, сравнение, обнаружение отклонений и т. д. [5].

3. Хранилища данных. Хранилище данных представляет собой репозитарий данных, собранных из нескольких источников (часто гетерогенных) и предназначенных для использования в целом по одной и той же единой схеме. Хранилище дает возможность анализировать данные из разных источников при помощи одних и тех же инструментов. Предположим, что наш интернет-магазин станет франшизой в Северной Америке. Многие видеомагазины, принадлежащие нашей компании, могут иметь разные базы данных и разные структуры. Если руководитель компании хочет получить доступ к данным из всех магазинов для принятия стратегических решений, определения будущих направлений, маркетинга и т. д., было бы более целесообразно хранить все данные на одном сайте с однородной структурой, позволяющей проводить интерактивный анализ. Другими словами, данные из разных хранилищ будут загружаться, очищаться, преобразовываться и интегрироваться вместе. Чтобы облегчить принятие решений и многомерные представления, хранилища данных обычно моделируются многомерной структурой данных [5]. На рис. 1 показан пример трехмерного подмножества структуры куба данных.

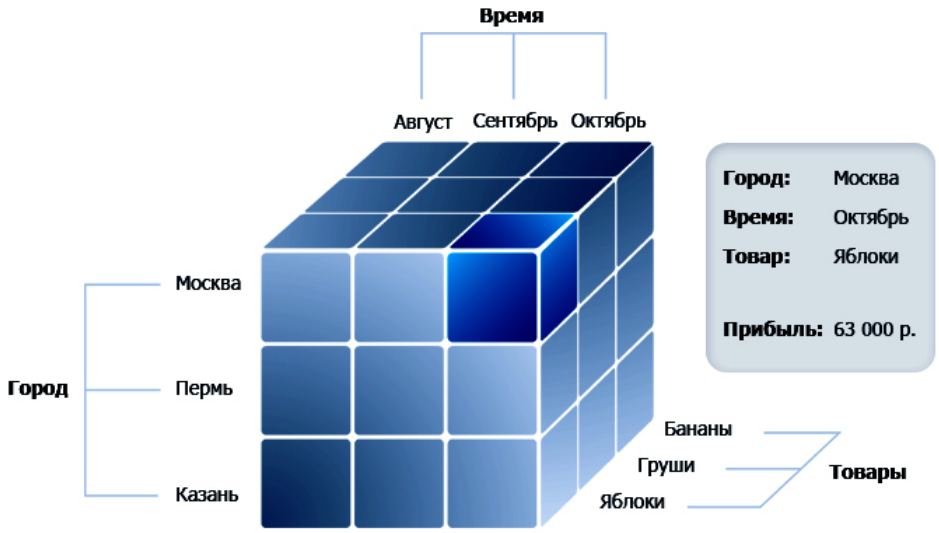


Рис. 1. Пример трехмерного подмножества структуры куба данных[1]

[1] *KAI* Development. URL:https://kaidev.ru/Pages/Article.aspx?p=Olap-About.

Куб данных дает сводные данные по трем измерениям: категория, время и город. Куб содержит ячейки, в которых хранятся значения некоторых агрегированных показателей (в данном случае прибыль от реализации конкретной категории товаров в определенном городе за определенный месяц), и специальные ячейки, в которых хранятся суммирования по измерениям. Каждое измерение куба данных содержит иерархию значений для одного атрибута.

Благодаря своей структуре, заранее вычисленным обобщенным данным, которые они содержат, и иерархическим значениям атрибутов их измерений, кубы данных хорошо подходят для быстрого интерактивного запроса и анализа данных на различных концептуальных уровнях, известных как оперативная аналитическая обработка (online analytical processing, OLAP). Операции OLAP позволяют осуществлять навигацию по данным на различных уровнях абстракции, таких как срез (рис. 2), вращение (рис. 3), детализация (рис. 4), консолидация (рис. 5) и т. д. [5].

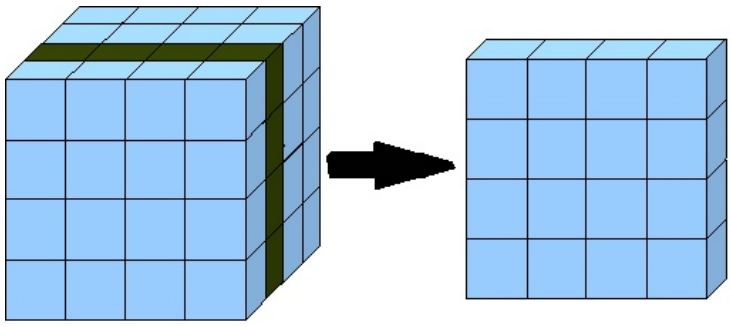


Рис. 2. Операции OLAP: срез

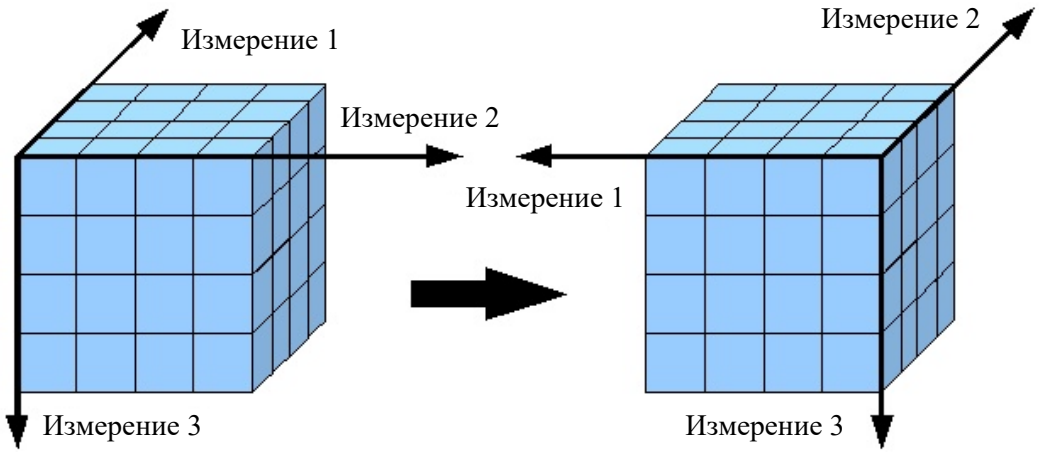


Рис. 3. Операции OLAP: вращение

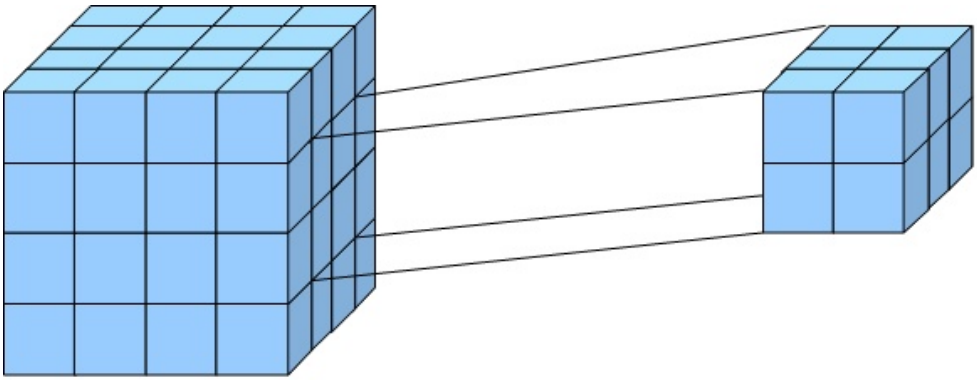


Рис. 4. Операции OLAP: детализация

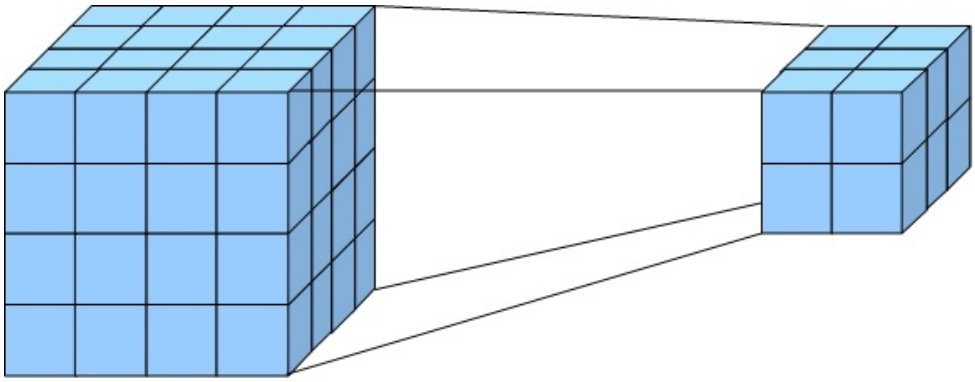


Рис. 5. Операции OLAP: консолидация

4. Транзакционные базы данных. Это набор записей, представляющих транзакции, каждая из которых имеет отметку времени, идентификатор и набор элементов. Связанные с файлами транзакции также могут быть описательными данными для элементов. Поскольку реляционные базы данных не допускают вложенных таблиц, т. е. набора данных в качестве значения атрибута, транзакции обычно хранятся в однородных файлах или в двух нормализованных таблицах транзакций, одна для транзакций и одна для элементов транзакций. Одним из типичных методов интеллектуального анализа данных является так называемый анализ рыночной корзины или ассоциативные правила, в которых изучаются ассоциации между элементами, встречающимися вместе или последовательно [5].

5. Мультимедийные базы данных. Мультимедийные базы данных включают видео, изображения, аудио и текстовые носители. Они могут храниться в расширенных объектно-реляционных или объектно-ориентированных базах данных или просто

в файловой системе. Мультимедиа характеризуется высокой размерностью, что делает интеллектуальный анализ данных еще более сложным. Интеллектуальный анализ данных из мультимедийных хранилищ может потребовать компьютерного зрения, компьютерной графики, интерпретации изображений и методов обработки естественного языка [5].

6. Пространственные базы данных. Помимо обычных данных, хранят географическую информацию, такую как карты и глобальное или региональное позиционирование [5]. Пространственные базы данных создают новые проблемы, для решения которых были разработаны такие алгоритмы, как интеллектуальный анализ пространственных шаблонов [9, с. 680], обнаружение пространственно-временных аномалий [9, с. 681], анализ пространственной автокорреляции [9, с. 681].

7. Базы данных временных рядов. Содержат связанные со временем данные, такие как данные фондового рынка или зарегистрированные действия (рис. 6). Эти базы обычно имеют непрерывный поток новых данных, что иногда вызывает необходимость сложного анализа в реальном времени. Интеллектуальный анализ данных в этом случае обычно включает изучение тенденций и корреляций между эволюциями различных переменных, а также прогнозирование тенденций и движений переменных во времени.

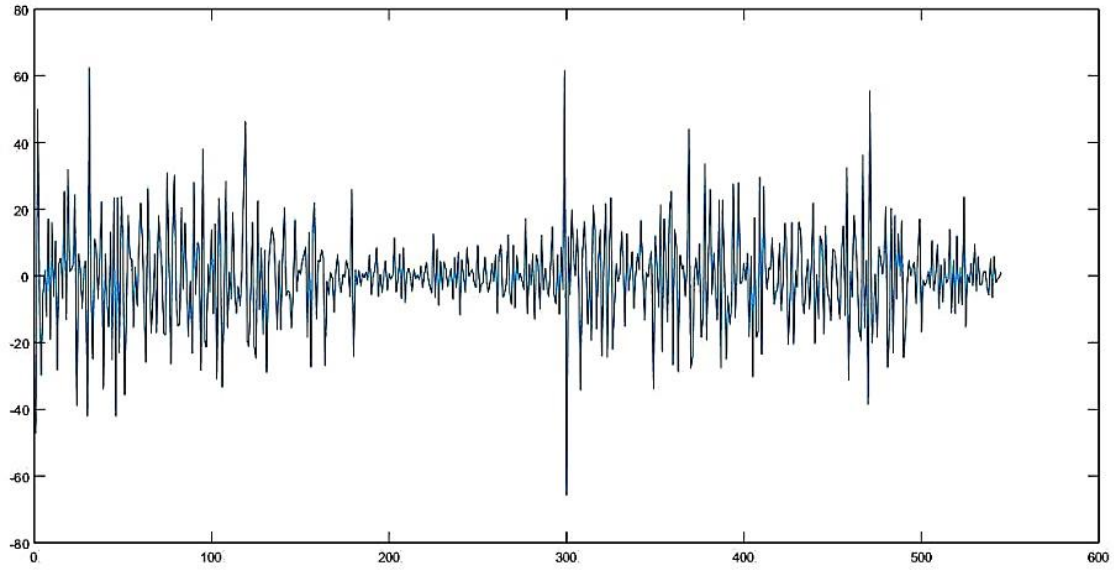


Рис. 6. Пример данных временных рядов

8. Всемирная паутина (WWW). Это самый разнородный и динамичный из доступных репозиториев. Множество авторов и издателей постоянно вносят свой вклад в его рост и изменение, и огромное количество пользователей ежедневно получает доступ к его ресурсам. Данные во Всемирной паутине организованы во взаимосвязанных документах. Эти документы могут быть текстовыми, аудио, видео, необработанными данными и даже приложениями. Концептуально Всемирная паутина состоит из трех основных компонентов:

– содержание Интернета, которое включает в себя доступные документы;

– структура сети, которая охватывает гиперссылки и отношения между документами;

– использование сети, описывающее, как и когда осуществляется доступ к ресурсам.

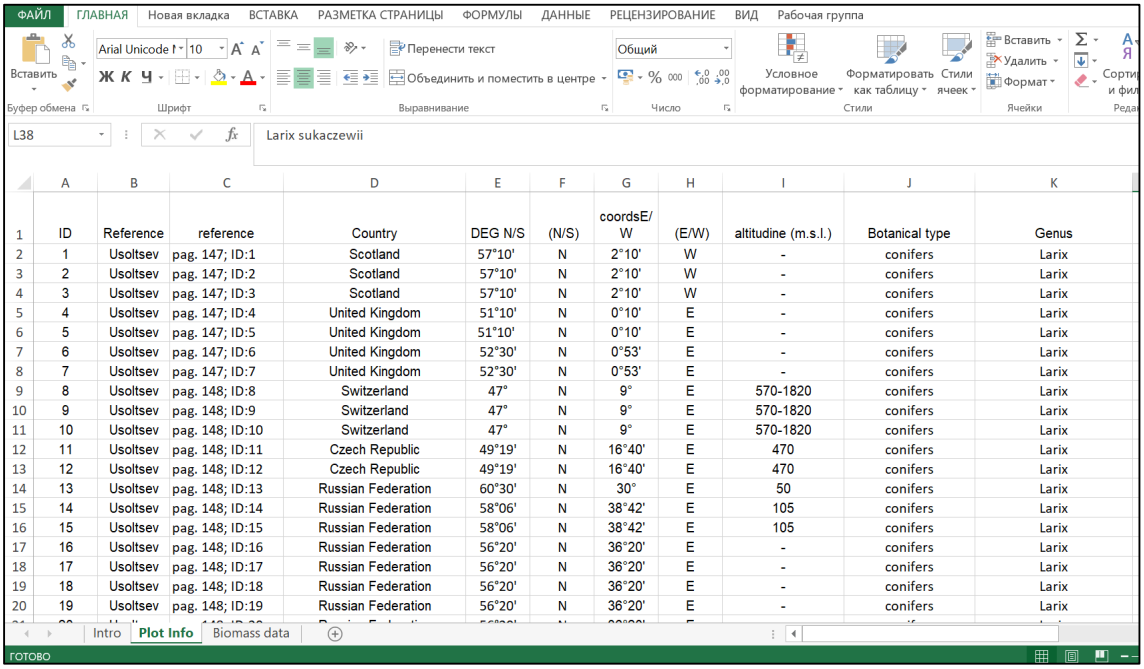
Можно добавить четвертое измерение, связанное с динамической природой или эволюцией документов. Интеллектуальный анализ данных во Всемирной паутине, или веб-майнинг, пытается решить все эти проблемы и часто делится на интеллектуальный анализ веб-контента, интеллектуальный анализ веб-структуры и интеллектуальный анализ веб-использования.

1.6. Инструменты формализации данных

В общем виде под формализацией данных подразумевается их приведение к стандартизированной форме. В области больших данных встречается много разных типов данных, для каждого из которых требуются свои инструменты и методы. Выделяются следующие основные категории данных [10, с. 21].

1. Структурированные данные. Зависят от модели данных и хранятся в фиксированном поле внутри записи. Соответственно, структурированные данные часто бывает удобно хранить в таблицах, базах данных или файлах Excel (рис. 7).

SQL является основным средством управления и обращения с запросами к данным, хранящимся в базах данных. Также иногда встречаются структурированные данные, которые достаточно трудно сохранить в традиционной реляционной базе дан-



L38 Larix sukaczewii

	ID	Reference	reference	Country	DEG N/S	(N/S)	coordsE/W	(E/W)	altitudine (m.s.l.)	Botanical type	Genus
2	1	Usoltsev	pag. 147; ID:1	Scotland	57°10'	N	2°10'	W	-	conifers	Larix
3	2	Usoltsev	pag. 147; ID:2	Scotland	57°10'	N	2°10'	W	-	conifers	Larix
4	3	Usoltsev	pag. 147; ID:3	Scotland	57°10'	N	2°10'	W	-	conifers	Larix
5	4	Usoltsev	pag. 147; ID:4	United Kingdom	51°10'	N	0°10'	E	-	conifers	Larix
6	5	Usoltsev	pag. 147; ID:5	United Kingdom	51°10'	N	0°10'	E	-	conifers	Larix
7	6	Usoltsev	pag. 147; ID:6	United Kingdom	52°30'	N	0°53'	E	-	conifers	Larix
8	7	Usoltsev	pag. 147; ID:7	United Kingdom	52°30'	N	0°53'	E	-	conifers	Larix
9	8	Usoltsev	pag. 148; ID:8	Switzerland	47°	N	9°	E	570-1820	conifers	Larix
10	9	Usoltsev	pag. 148; ID:9	Switzerland	47°	N	9°	E	570-1820	conifers	Larix
11	10	Usoltsev	pag. 148; ID:10	Switzerland	47°	N	9°	E	570-1820	conifers	Larix
12	11	Usoltsev	pag. 148; ID:11	Czech Republic	49°19'	N	16°40'	E	470	conifers	Larix
13	12	Usoltsev	pag. 148; ID:12	Czech Republic	49°19'	N	16°40'	E	470	conifers	Larix
14	13	Usoltsev	pag. 148; ID:13	Russian Federation	60°30'	N	30°	E	50	conifers	Larix
15	14	Usoltsev	pag. 148; ID:14	Russian Federation	58°06'	N	38°42'	E	105	conifers	Larix
16	15	Usoltsev	pag. 148; ID:15	Russian Federation	58°06'	N	38°42'	E	105	conifers	Larix
17	16	Usoltsev	pag. 148; ID:16	Russian Federation	56°20'	N	36°20'	E	-	conifers	Larix
18	17	Usoltsev	pag. 148; ID:17	Russian Federation	56°20'	N	36°20'	E	-	conifers	Larix
19	18	Usoltsev	pag. 148; ID:18	Russian Federation	56°20'	N	36°20'	E	-	conifers	Larix
20	19	Usoltsev	pag. 148; ID:19	Russian Federation	56°20'	N	36°20'	E	-	conifers	Larix

Рис. 7. Таблица Excel как пример структурированных данных

ных (один из примеров — иерархические данные, например генеалогическое дерево). Однако мир не состоит из структурированных данных; просто это представление удобно для человека и машин. Чаще реальные данные хранятся в неструктурированном виде [10, с. 21].

2. Неструктурированные данные трудно подогнать под конкретную модель, потому что их содержимое зависит от контекста или имеет переменный характер. Один из примеров неструктурированных данных — обычные сообщения электронной почты (рис. 8). Хотя сообщение содержит структурированные элементы (отправитель, заголовок, тело), одни и те же задачи могут решаться множеством разных способов, например, существует бесчисленное количество вариантов упоминания конкретного человека в сообщениях. Проблема дополнительно усложняется существованием тысяч языков и диалектов. Сообщение электронной почты, написанное человеком, также является идеальным примером данных на естественном языке [10, с. 22].

3. Данные на естественном языке составляют особую разновидность неструктурированных данных; их обработка достаточно сложна, потому что требует знания как лингвистики, так и специальных методов data science. Сообщество обработки дан-

Рис. 8. Сообщение электронной почты как пример неструктурированных данных и данных на естественном языке

ных на естественном языке добилось успеха в области распознавания сущностей, распознавания тематических областей, обобщения, завершения текста и анализа эмоциональной окраски, но модели, адаптированные для одной предметной области, плохо обобщаются для других областей. Даже самые современные методы не помогут расшифровать смысл произвольного фрагмента текста. И этот факт вряд ли кого-то удивит: у людей также возникают проблемы с восприятием естественного языка. Он неоднозначен по своей природе. Сама концепция смысла выглядит спорно. Два человека слушают один разговор; вынесут ли они одинаковый смысл из него? Даже смысл отдельных слов может изменяться в зависимости от настроения говорящего [10, с. 22–23].

4. Машинные данные. К машинным данным относится информация, автоматически генерируемая компьютером, процессом, приложением или устройством без вмешательства человека. Машинные данные становятся одним из основных источников информации, и ситуация вряд ли изменится. По данным Wikibon рыночная стоимость промышленного Интернета (термин, предложенный компанией Frost&Sullivan для обозначения совокупности сложного физического оборудования с сетевыми датчиками и программным обеспечением) к 2020 г. должна была соста-

вить приблизительно 540 млрд долл. А количество узлов, по оценкам IDC (International Data Corporation), к этому времени должно было в 26 раз превысить численность населения. Эта сеть часто называется Интернетом вещей.

Анализ машинных данных из-за их громадных объемов и скоростей сильно зависит от инструментов с высокой масштабируемостью. К примерам машинных данных относятся журналы веб-серверов, записи детализации звонков, журналы сетевых событий и телеметрии (рис. 9). Эти машинные данные хорошо укладываются в структуру классической базы данных. Это не лучший формат для данных с высокой степенью связности или «сетевых» данных, в которых достаточно значимую роль играют отношения между сущностями [10, с. 23–24].

27.05.2015 11:38:26...	Configuration option 'show advanced options' changed from 0 to 1. Run the RECONFIGURE statement to install.	spid56	SQL Server
27.05.2015 11:38:26...	Configuration option 'show advanced options' changed from 1 to 0. Run the RECONFIGURE statement to install.	spid56	SQL Server
27.05.2015 11:38:29...	Configuration option 'show advanced options' changed from 0 to 1. Run the RECONFIGURE statement to install.	spid59	SQL Server
27.05.2015 11:38:29...	Configuration option 'show advanced options' changed from 1 to 0. Run the RECONFIGURE statement to install.	spid59	SQL Server
27.05.2015 11:38:29...	SQL Trace ID 18 was started by login "SoftpointPerfomanceExpertLicenceUser".	spid59	SQL Server
27.05.2015 11:38:31...	SQL Trace stopped. Trace ID = '18'. Login Name = 'SoftpointPerfomanceExpertLicenceUser'.	spid59	SQL Server
27.05.2015 11:38:34...	SQL Trace ID 18 was started by login "sa".	spid56	SQL Server
27.05.2015 11:38:34...	SQL Trace ID 19 was started by login "sa".	spid76	SQL Server
27.05.2015 11:39:34...	SQL Trace stopped. Trace ID = '18'. Login Name = 'sa'.	spid56	SQL Server

Рис. 9. Пример машинных данных

5. Графовые или сетевые данные. Термин «графовые данные» может сбить с толку, потому что любые данные могут быть представлены в виде графа. В данном случае имеется в виду понятие графа из математической теории графов — математическая структура для моделирования попарных отношений между объектами. Вкратце, в графовых, или сетевых, данных особое внимание уделяется связям или смежности объектов. Графовые структуры данных используют узлы, ребра и свойства для представления и хранения графических данных. Графовые данные естественным образом подходят для представления социальных сетей, а их структура позволяет вычислять такие специфические метрики, как влияние участников и кратчайший путь между двумя людьми.

Примеры графовых данных встречаются на многих веб-сайтах социальных сетей (рис. 10). Например, в LinkedIn можно увидеть, кого вы знаете в той или иной компании. Ваш список читателей в Твиттере также является примером графовых дан-

ных. Сила и мощь связанных данных проявляется при анализе нескольких перекрывающихся графов, построенных на одних и тех же узлах. Например, представьте, что ребра обозначают «друзей» на Facebook. А теперь возьмем другой граф с теми же людьми, но связывающий коллег по бизнесу через LinkedIn, и третий граф, основанный на интересе к фильмам на Netflix. Наложение этих трех графов позволит получить ответы на многие вопросы. Для хранения графовых данных используются графовые базы данных, а для построения запросов к ним — такие специализированные языки запросов, как SPARQL.

Рис. 10. Пример структуры графовых данных[1]

Работа с графовыми данными создает специфические проблемы, причем для компьютера эта задача становится еще сложнее [10, с. 24–25].

6. Аудио, видео и графика — типы данных, ставящие непростые задачи перед специалистами data science (ученые по данным). Задачи, тривиальные с точки зрения человека (например, распознавание объекта на картинке), оказываются сложными для компьютера. В 2014 г. компания MLBAM (Major League Baseball Advanced Media) объявила, что объем записываемых видеоматериалов для одного бейсбольного матча будет увеличен приблизительно до 7 Тбайт с целью проведения оперативного анализа. Высокоскоростные камеры на стадионах записывают движения мяча и спортсменов для того, например, чтобы вычис-

[1] *Конференция* Graph+AI World 2020 — графовые алгоритмы и машинное обучение / Хабр. URL: https://m.habr.com/ru/company/fgts/blog/520708/.

лять в реальном времени траекторию движения защитника. Недавно компании DeepMind удалось создать алгоритм, который способен обучаться играть в видеоигры. Алгоритм получает на входе содержимое экрана и учится интерпретировать эти данные в сложном процессе глубокого обучения. Компания Google приобрела DeepMind для разработки искусственного интеллекта. Алгоритм обучения получает данные, генерируемые компьютерной игрой, т. е. потоковые данные [10, с. 25–26].

7. Потоковые данные могут принимать почти любую из перечисленных форм, однако у них имеется одно дополнительное свойство. Данные поступают в систему при возникновении некоторых событий, а не загружаются в хранилище большими массивами. И хотя формально они не являются отдельной разновидностью данных, мы выделяем их в особую категорию.

Примерами потоковых данных могут служить раздел «Что происходит?» в Твиттере, прямые трансляции спортивных и музыкальных мероприятий и данные биржевых котировок [10, с. 26].

Процесс сбора данных подвержен ошибкам; в этой фазе исследователь повышает качество данных и готовит их к последующему использованию. Эта фаза состоит из трех подфаз [10, с. 27]:

– очистка данных удаляет некорректные значения из источника данных и устраняет расхождения между источниками;

– интеграция данных расширяет возможности посредством объединения информации из нескольких источников;

– преобразование данных гарантирует, что данные находятся в подходящем формате для использования в ваших моделях.

В настоящее время существует много разных инструментариев и инфраструктур для больших данных. В них легко запутаться, потому что новые технологии появляются очень быстро. Экосистема больших данных может быть разбита на группы по технологиям с похожими целями и функциональностью. Специалисты data science применяют много технологий, но не все сразу. На рис. 11 показаны компоненты экосистемы больших данных и место разных технологий в этой структуре [10, с. 28]. Рассмотрим некоторые компоненты более подробно.

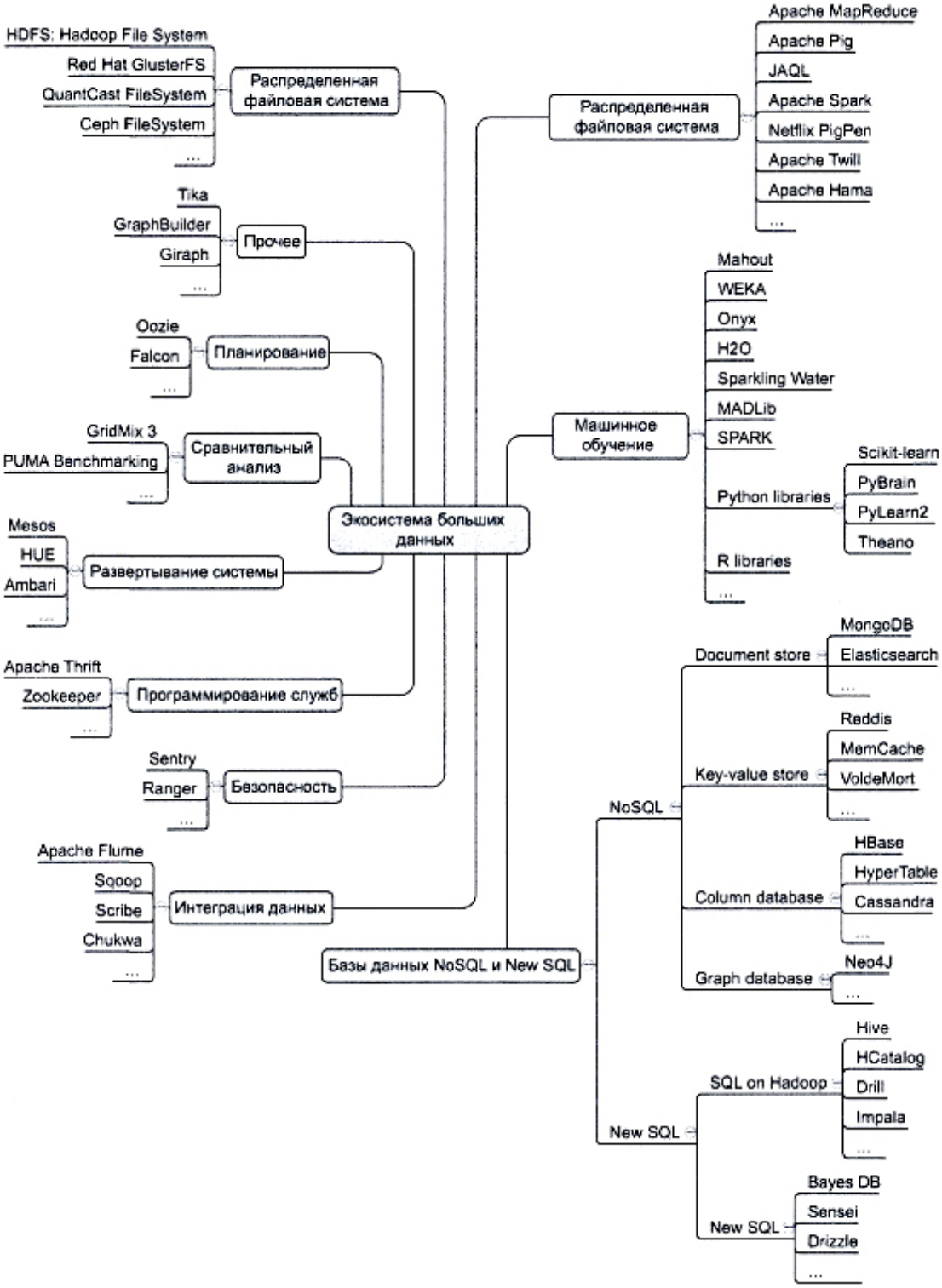


Рис. 11. Технологии больших данных по основным категориям [10, с. 29]

Распределенные файловые системы. Распределенная файловая система похожа на обычную файловую систему, но в отличие от последней она функционирует на нескольких серверах

сразу. Используя эти системы, можно выполнять почти все те же действия, что и в обычных файловых системах. В основе любой файловой системы лежат такие действия, как запись, хранение, чтение и удаление данных, а также реализация средств безопасности файлов. Распределенные файловые системы имеют такую же функциональность и обладают при этом рядом важных преимуществ [10, с. 30]:

– они способны хранить файлы, размер которых превышает размер диска отдельного компьютера;

– файлы автоматически реплицируются на нескольких серверах для создания избыточности или выполнения параллельных операций, при этом все сложности технической реализации этих действий незаметны для пользователя;

– распределенная файловая система легко масштабируется — пользователь не ограничен объемом памяти или дискового пространства одного сервера.

Возможность масштабирования является важной характеристикой файловой системы. Ранее масштабирование осуществлялось переводом всех систем на сервер с большим объемом памяти и дискового пространства и более быстрым процессором (вертикальное масштабирование). В настоящее время в распределенных системах появляется возможность дополнительного использования соседнего по уровню сервера с унифицированными характеристиками (горизонтальное масштабирование). Благодаря такой возможности потенциал масштабирования становится практически безграничным. Наиболее популярной распределенной файловой системой является Hadoop File System (HDFS). Она представляет собой реализацию Google File System с открытым кодом. Эта система чаще всего применяется на практике. Существуют также и другие распределенные файловые системы: Red Hat Cluster File System, Ceph File System, Tachyon File System и др. [10, с. 30].

Технологии распределенного программирования. После того как данные сохранены в распределенной файловой системе, следует процесс их использования. Важнейший аспект работы с распределенной файловой системой заключается в том, что более рациональным является не перемещение данных к программе, а наоборот, перемещение программы к данным [10, с. 30].

Инфраструктура интеграции данных. После создания распределенной файловой системы возникает необходимость добавления данных или перемещения данных из одного источника в другой. В подобных случаях используются такие инфраструктуры интеграции данных, как Apache Sqoop и Apache Flume [10, с. 31].

Базы данных NoSQL. Для хранения огромных объемов данных требуется программное обеспечение, специализирующееся на управлении этими данными и формировании запросов к ним. Традиционно в этой области использовались реляционные базы данных, такие как Oracle SQL, MySQL. Sybase IQ и др. Появились и новые типы СУБД, объединенные в категорию NoSQL. Существующие разновидности баз данных можно разделить на следующие типы [10, с. 32–33]:

– столбцовые базы данных (column database) — данные организуются в столбцы, что позволяет алгоритмам существенно повышать скорость обработки запросов; табличные структуры продолжают играть важную роль в обработке данных;

– хранилища документов (document store) — хранилища документов, не использующие таблицы, но хранящие полную информацию о документе; их особенностью является чрезвычайно гибкая схема данных;

– хранилища «ключ-данные» (key-value store) — данные не хранятся в таблицах; каждому отдельному значению ставится в соответствие ключ, а не «координаты» этого значения в таблице; такое решение обеспечивает хорошее масштабирование, но затрудняет разработку базы данных;

– SQL в Hadoop — пакетные запросы в Hadoop пишутся на SQL-подобном языке, во внутренней реализации которого используется инфраструктура отображения-свертки (Map-reduce);

– обновленный SQL (New SQL) — этот тип сочетает масштабируемость баз данных NoSQL с преимуществами реляционных баз данных; используется интерфейс SQL и реляционная модель данных.

Графовые базы данных (graph database). Табличный формат представления данных является оптимальным не для всех задач. Некоторые задачи могут быть более естественно представлены с помощью графовых моделей, а используемые ими

данные размещены в графовых базах данных. Классический пример такого рода — социальная сеть.

Инструменты планирования. Инструменты планирования упрощают автоматизацию повторяющихся операций и запуск заданий по событиям (например, при появлении нового файла в папке). Эти инструментальные средства имеют аналоги в традиционных программах, но разрабатываются специально для больших данных. Например, такие инструменты могут запускать задачу Map-reduce при появлении нового набора данных в каталоге [10, с. 33].

Выводы по главе 1

Понятия «данные» и «информация» тесно связаны. Под данными обычно понимают информацию, представленную в конкретных формах, которые адекватны возможным (ожидаемым) процессам ее обработки.

В зависимости от целей использования при оценке информации учитываются ее синтаксический, семантический и прагматический аспекты.

Четкого определения «больших данных» не существует. В рамках данного учебного пособия под «большими данными» понимается совокупность данных, технологии их обработки и методы поиска.

Источниками больших данных являются деловые операции, научные данные, медицинские и личные данные, видеонаблюдение и фотографии и многие другие.

Основные проблемы в сфере обработки больших данных:

– получение информации об объеме и достоверности имеющихся данных;

– разрозненность данных;

– неточность данных;

– конфиденциальность личных данных;

– потеря информации;

– затраты на обработку.

Основные способы хранения больших данных — однородные файлы, реляционные и нереляционные базы данных,

хранилища данных, транзакционные базы данных, мультимедийные базы данных, пространственные базы данных, базы данных временных рядов, Всемирная паутина. Предварительная обработка и формализация данных включает этапы их очистки, интеграции и преобразования.

Вопросы для самоконтроля

1. Что такое данные? Что такое информация? Что такое знания? Чем данные отличаются от информации?
2. Раскройте понятие «большие данные». Три шага к новому способу анализа информации.
3. Назовите основные источники больших данных. Назовите причины появления больших данных.
4. Какие проблемы возникают в связи с необходимостью хранения, обработки и анализа больших данных?
5. Назовите и опишите основные способы хранения и представления больших данных. Что такое хранилище данных? Какие операции с кубами данных существуют?
6. Каковы основные категории данных? Назовите и опишите основные инструменты и этапы формализации.

Глава 2

Традиционные модели и типы данных

2.1. Иерархическая модель

Первые иерархические и сетевые СУБД были созданы в начале 1960-х гг. Причиной послужила необходимость управления миллионами записей (связанных друг с другом иерархическим образом), например, при информационной поддержке лунного проекта Аполлон [11, с. 47].

Иерархическая структура должна удовлетворять условиям:

– существует только один корневой узел;

– узел содержит один или несколько атрибутов, описывающих объект в данном узле;

– доступ к порожденным узлам возможен только через исходный узел (существует только один путь доступа к каждому узлу).

В БД может быть несколько деревьев, порожденных различными корневыми записями.

Корневая запись должна содержать ключ. Ключи некорневых записей должны быть уникальны только в пределах своего уровня иерархии того дерева, которому они принадлежат. Каждая запись идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой по иерархическому пути, ведущему к данной записи. Основная единица обработки — запись [12, с. 43].

Иерархическая модель данных фактически является частным случаем сетевой модели.

Компоненты базы данных, основанной на иерархической модели, могут быть представлены в виде схемы (рис. 12).

Рис. 12. Компоненты баз данных, основанных на иерархической модели

Если никакой тип записи прямо или опосредованно не может быть владельцем самого себя, то соответствующую схему БД можно представить в виде ациклического графа — совокупности деревьев.

Более конструктивное определение дерева можно дать следующим образом.

Дерево — это конечное множество узлов T, таких, что [12, с. 44–45]:

1) имеется один специально выделенный узел, называемый корнем дерева;

2) остальные узлы (исключая корень) содержаться в $m \geq 0$ попарно не пересекающихся множествах T_1, T_2, …, T_m, каждое из которых в свою очередь является деревом. Деревья T_1, T_2, …, T_m называются поддеревьями данного корня.

Совокупность деревьев называют лесом.

Иерархическая древовидная структура строится из узлов и ветвей. Компоненты базы данных, основанной на иерархической модели хранения данных, показаны на рис. 13 в виде древовидной структуры.

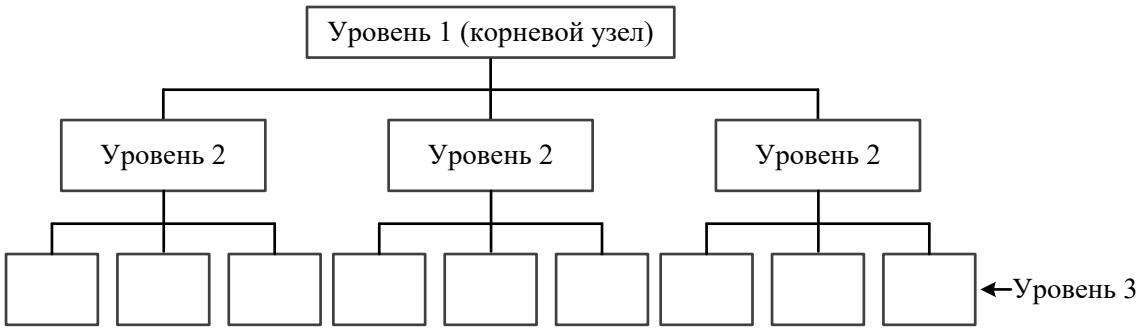


Рис. 13. Древовидная структура компонентов баз данных, основанных на иерархической модели хранения данных

Организация данных в СУБД иерархического типа определяется в терминах: элемент (атрибут), запись (группа), групповое отношение, база данных [12, с. 47].

Атрибут (элемент данных) — наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Элемент данных также часто называют полем.

Запись — именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Тип записи определяется составом ее атрибутов. Экземпляр записи — конкретная запись с конкретным значением элементов

Групповое отношение — иерархическое отношение между записями двух типов. Родительская запись (владелец группового отношения) называется исходной записью, а дочерние записи (члены группового отношения) — подчиненными. Иерархическая база данных может хранить только такие древовидные структуры.

Корневая запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках группового отношения.

Каждая запись идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой по иерархическому пути.

Рис. 14. Групповое отношение кафедра — сотрудник

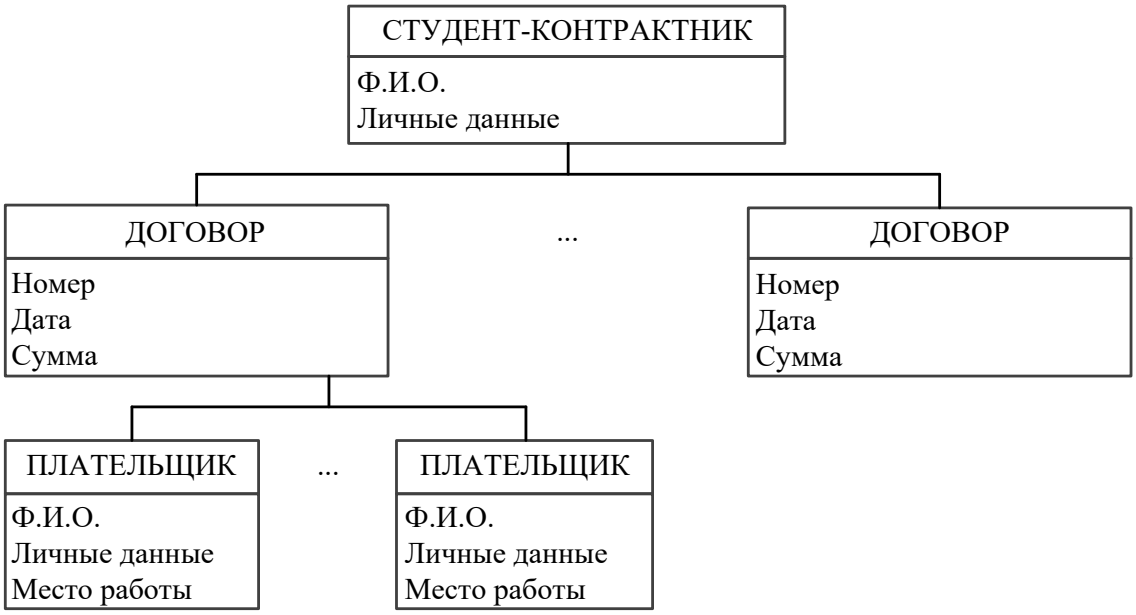


Рис. 15. Групповое отношение студент — договор — плательщик

Рис. 16. Дополнительное групповое отношение студент — договор — плательщик

Рассмотрим следующую модель данных учебного заведения (рис. 14–16): в заведении находятся кафедры, на которых работают сотрудники. На каждой кафедре может работать несколько сотрудников, но сотрудник не может работать более чем на одной кафедре.

Для информационной системы управления персоналом заведения необходимо создать групповое отношение, состоящее из родительской записи КАФЕДРА (НАИМЕНОВАНИЕ КАФЕДРЫ, КОЛИЧЕСТВО РАБОТНИКОВ) и дочерней записи СОТРУДНИК (Ф.И.О., ДОЛЖНОСТЬ, ЛИЧНЫЕ ДАННЫЕ). Это отношение показано на рис. 14 (для простоты полагается, что имеются только две дочерние записи).

Для автоматизации учета контрактов со студентами (оплата за обучение) необходимо создание еще одной иерархической структуры: студент — договоры с ним — плательщики по договору. Это дерево будет включать записи (рис. 15) СТУДЕНТ-КОНТРАКТНИК (Ф.И.О., ЛИЧНЫЕ ДАННЫЕ), ДОГОВОР (НОМЕР, ДАТА, СУММА), ПЛАТЕЛЬЩИК (Ф.И.О., ЛИЧНЫЕ ДАННЫЕ, МЕСТО РАБОТЫ).

Из этого примера видны недостатки иерархических БД.

Иерархическая модель реализует отношение между исходной и дочерней записью по схеме 1:*N*, т. е. одной родительской записи может соответствовать любое число дочерних. Допустим теперь, что плательщик может принимать участие более чем в одном договоре (возникает связь типа *M*:*N*). В этом случае в базу данных необходимо ввести еще одно групповое отношение, в котором ПЛАТЕЛЬЩИК будет являться исходной записью, а ДОГОВОР — дочерней (рис. 16). Таким образом, мы вынуждены дублировать информацию.

2.2. Сетевая модель

На разработку этого стандарта большое влияние оказал американский ученый Ч. Бахман. Основные принципы сетевой модели данных были разработаны в середине 1960-х гг., эталонный вариант сетевой модели данных описан в отчетах рабочей группы по языкам баз данных CODASYL (COnference on DAta

SYstem Languages, 1971 г.). Этой же организацией в 1975 г. был определен стандарт сетевой модели, а также базовые понятия модели и формальный язык описания [11, с. 51].

Сетевая модель СУБД во многом подобна иерархической: если в иерархической модели для каждого сегмента записи допускается только один входной сегмент при *N* выходных, то в сетевой модели для сегментов допускается несколько входных сегментов наряду с возможностью наличия сегментов без входов с точки зрения иерархической структуры.

Графическое изображение структуры связей сегментов такого типа моделей представляет собой сеть. Сегменты данных в сетевых БД могут иметь множественные связи с сегментами старшего уровня. При этом направление и характер связи в сетевых БД не являются столь очевидными, как в случае иерархических БД. Поэтому имена и направление связей должны идентифицироваться при описании БД.

Таким образом, под сетевой СУБД понимается система, поддерживающая сетевую организацию: любая запись, называемая записью старшего уровня, может содержать данные, которые относятся к набору других записей, называемых записями подчиненного уровня. Возможно обращение ко всем записям в наборе начиная с записи старшего уровня. Обращение к набору записей реализуется по указателям.

Сетевая модель данных состоит из множества записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также имеет вид 1:*N*.

Основное различие сетевой и иерархической моделей состоит в том, что в сетевой модели запись может быть членом более чем одного группового отношения. Согласно этой модели каждое групповое отношение именуется и проводится различие между его типом и экземпляром. Тип группового отношения задается его именем и определяет свойства общие для всех экземпляров данного типа. Экземпляр группового отношения представляется записью-владельцем и множеством (возможно пустым) подчиненных записей. При этом имеется следующее ограничение: экземпляр записи не может быть членом двух экземпляров групповых отношений одного типа.

Каждый экземпляр группового отношения характеризуется следующими признаками [12, с. 55–57]:

1) способ упорядочения подчиненных записей:

– произвольный;

– хронологический (очередь);

– обратный хронологический (стек);

– сортированный.

Если запись объявлена подчиненной в нескольких групповых отношениях, то в каждом из них может быть назначен свой способ упорядочивания;

2) режим включения подчиненных записей:

– автоматический — невозможно занести в БД запись без того, чтобы она была сразу же закреплена за неким владельцем;

– ручной — позволяет запомнить в БД подчиненную запись и не включать ее немедленно в экземпляр группового отношения. Эта операция позже инициируется пользователем;

3) режим исключения.

Принято выделять три класса членства подчиненных записей в групповых отношениях:

1) фиксированное. Подчиненная запись жестко связана с записью владельца, и ее можно исключить из группового отношения только удалением. При удалении записи-владельца все подчиненные записи автоматически тоже удаляются;

2) обязательное. Допускается переключение подчиненной записи на другого владельца, но невозможно ее существование без владельца. Для удаления записи-владельца нужно, чтобы она не имела подчиненных записей с обязательным членством;

3) необязательное. Можно исключить запись из группового отношения, но сохранить ее в базе данных без прикрепления ее к другому владельцу. При удалении записи-владельца ее подчиненные записи — необязательные члены сохраняются в базе, не участвуя более в групповом отношении такого типа.

Организация данных определяется в терминах:

– элемент данных;

– агрегат данных;

– запись;

– ключ;

– набор данных.

Элемент данных — то же, что и в иерархической модели, т. е. минимальная информационная единица, доступная пользователю с использованием СУБД.

Агрегат данных — следующий уровень обобщения в модели. В модели определены агрегаты двух типов: вектор и повторяющаяся группа.

Агрегат данных имеет имя, и в системе допустимо обращение к агрегату по имени. Агрегат типа вектор соответствует линейному набору элементов данных. Например, агрегат «Адрес» может быть представлен следующим образом (рис. 17).

Адрес				
Страна	Город	Улица	Дом	Квартира

Рис. 17. Пример агрегата типа вектор

Агрегат типа повторяющаяся группа соответствует совокупности векторов данных. Например, агрегат «Зарплата» может соответствовать типу повторяющаяся группа с числом повторений 12 (рис. 18).

Зарплата	
Месяц	Сумма

Рис. 18. Пример агрегата типа повторяющаяся группа

Запись — это совокупность агрегатов или элементов данных, моделирующая некоторый класс объектов реального мира. Понятие записи соответствует понятию «сегмент» в иерархической модели. Для записи, так же как и для сегмента, вводятся понятия типа записи и экземпляра записи.

Ключ — некоторая совокупность элементов, идентифицирующих запись.

Набором называется двухуровневый граф, связывающий отношением «один-ко-многим» два типа записи.

В качестве примера рассмотрим табл. 1 и рис. 19, на основе которой организуем два набора и определим связь между ними.

Таблица 1

Данные об учебном процессе

Преподаватель	Группа	День недели	№ пары	Аудитория	Дисциплина
Иванов	4306	Понедельник	1	22-13	КИД
Иванов	4307	Понедельник	2	22-13	КИД
Карпова	4307	Вторник	2	22-14	БЗ и ЭС
Карпова	4309	Вторник	4	22-14	БЗ и ЭС
Карпова	4305	Вторник	1	22-14	БД
Смирнов	4306	Вторник	3	23-07	ГВП
Смирнов	4309	Вторник	4	23-07	ГВП

Рис. 19. Модель учебного процесса

Экземпляров набора «Ведет занятия в» будет 3 (по числу преподавателей), экземпляров набора «Занимается у» будет 4 (по числу групп). На рис. 20 представлены взаимосвязи экземпляров данных наборов.

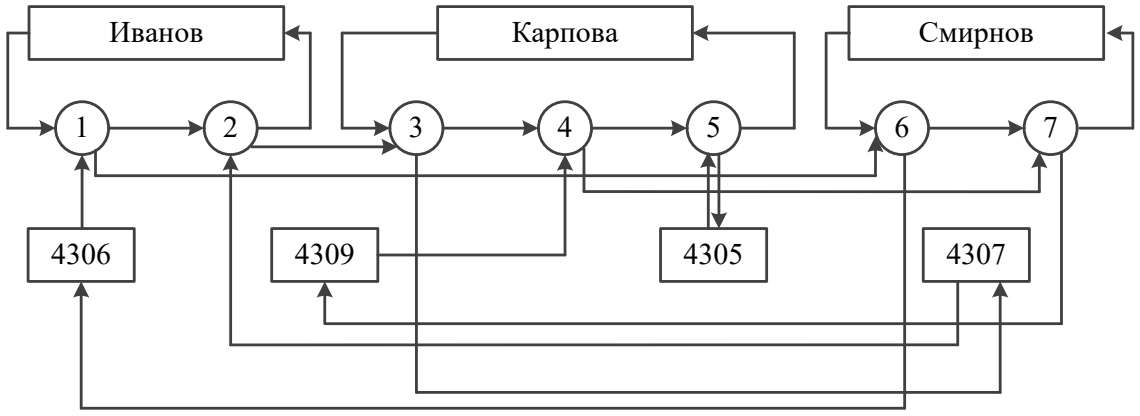


Рис. 20. Пример взаимосвязи экземпляров двух наборов

Все операции манипулирования данными в сетевой модели делятся на навигационные и операции модификации.

Навигационные операции осуществляют перемещение по БД путем прохождения по связям, которые поддерживаются в схеме БД. В этом случае результатом является новый единичный объект, который получает статус текущего объекта.

Операции модификации осуществляют добавление как новых экземпляров отдельных типов записей, так и экземпляров новых наборов, удаление экземпляров записей и наборов, модификацию отдельных составляющих внутри конкретных экземпляров записей.

Средства модификации данных сведены в табл. 2.

Таблица 2

Операторы манипулирования данными в сетевой модели

Операция	Назначение
READY	Обеспечение доступа данного процесса или пользователя к БД (сходна по смыслу с операцией открытия файла)
FINISH	Окончание работы с БД
FIND	Группа операций, устанавливающих указатель найденного объекта на текущий объект
GET	Передача найденного объекта в рабочую область. Допустимо только после оператора FIND
STORE	Помещение в БД записи, сформированной в рабочей области
CONNECT	Включение текущей записи в текущий экземпляр набора
DISCONNECT	Исключение текущей записи из текущего экземпляра набора
MODIFY	Обновление текущей записи данными из рабочей области пользователя
ERASE	Удаление экземпляра текущей записи

Ограничения целостности — как и в иерархической модели, обеспечивается только поддержание целостности по ссылкам (владелец отношения — член отношения).

2.3. Реляционная модель

Реляционная модель появилась в 1970 г. (Е. Ф. Кодд). Более ранними и основными используемыми в СУБД для универсальных ЭВМ являются сетевая и иерархическая модели данных. Определения этих моделей базируются на терминологии, предложенной профессиональной организацией в ряде отчетов, опубликованных в 1970 г. [12, с. 63].

В течение долгого времени реляционный подход рассматривался как удобный формальный аппарат анализа баз данных, не имеющий практических перспектив, так как его реализация требовала слишком больших машинных ресурсов. Только с появлением персональных ЭВМ реляционные и близкие к ним системы стали распространяться, и в настоящее время реляционная модель является фактическим стандартом, на который ориентируются большинство современных коммерческих СУБД.

Реляционная модель состоит из трех частей:

– структурной;

– целостной;

– манипуляционной.

Структурная часть описывает, какие объекты рассматриваются реляционной моделью. Постулируется, что единственной структурой данных, используемой в реляционной модели, являются нормализованные *n*-арные отношения.

Целостная часть описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и целостность внешних ключей.

Манипуляционная часть описывает два эквивалентных способа манипулирования реляционными данными — реляционную алгебру и реляционное исчисление.

Реляционная модель ориентирована на организацию данных в виде двумерных таблиц. Каждая реляционная таблица представляет собой двумерный массив и обладает свойствами:

– каждый элемент таблицы — один элемент данных, повторяющиеся группы отсутствуют;

– все столбцы в таблице однородные, т. е. все элементы в столбце имеют одинаковый тип (числовой, символьный и т. д.) и длину;

– каждый столбец имеет уникальное имя;

– одинаковые строки в таблице отсутствуют;

– порядок следования строк и столбцов может быть произвольным.

Таблица такого рода называется отношением. База данных, построенная с помощью отношений, называется реляционной базой данных.

Отношения представлены в виде таблиц, строки которых соответствуют кортежам или записям, а столбцы — атрибутам отношений, доменам, полям.

Поле, каждое значение которого однозначно определяет соответствующую запись, называется простым ключом (ключевым полем). Если записи однозначно определяются значениями нескольких полей, то такая таблица базы данных имеет составной ключ.

Чтобы связать две реляционные таблицы, необходимо ключ первой таблицы ввести в состав ключа второй таблицы (возможно совпадение ключей); в противном случае нужно ввести в структуру первой таблицы внешний ключ — ключ второй таблицы.

Существует два подхода к проектированию реляционной базы данных [12, с. 67−69]:

– первый подход заключается в том, что на этапе концептуального проектирования создается не концептуальная модель данных, а непосредственно реляционная схема базы данных, состоящая из определений реляционных таблиц, подвергающихся нормализации;

– второй подход основан на механическом преобразовании функциональной модели, созданной ранее, в нормализованную реляционную модель. Этот подход чаще всего используется при проектировании больших, сложных схем баз данных для корпоративных информационных систем.

Главной структурной единицей в реляционной модели данных являются не отдельные записи-кортежи, а множества кортежей — отношения.

Отношения обладают следующими свойствами:

– не содержат кортежей-дубликатов;

– кортежи отношений не упорядочены;

– атрибуты отношений не упорядочены;

– значения всех атрибутов атомарны (в них не присутствуют составные атрибуты), такие отношения называют представленными в первой нормальной форме (в виде плоских таблиц).

Отношения удобно представлять в виде таблиц. На рис. 21 представлена таблица (отношение степени 5), содержащая некоторые сведения о работниках гипотетического предприятия.

Строки таблицы соответствуют кортежам. Каждая строка фактически представляет собой описание одного объекта реального мира (в данном случае работника), характеристики которого содержатся в столбцах. Можно провести аналогию между элементами реляционной модели данных и элементами модели «сущность-связь». Реляционные отношения соответствуют наборам сущностей, а кортежи — сущностям. Поэтому, так же как и в модели «сущность-связь», столбцы в таблице, представляющей реляционное отношение, называют атрибутами.

Рис. 21. Основные компоненты реляционного отношения

Каждый атрибут определен на домене, поэтому домен можно рассматривать как множество допустимых значений данного атрибута.

Несколько атрибутов одного отношения и даже атрибуты разных отношений могут быть определены на одном и том же домене. В примере, показанном на рис. 21, атрибуты «Оклад» и «Премия» определены на домене «Деньги». Поэтому понятие домена имеет семантическую нагрузку: данные можно считать сравнимыми только тогда, когда они относятся к одному домену. Таким образом, в рассматриваемом нами примере сравнение атрибутов «Табельный номер» и «Оклад» является семантически некорректным, хотя они и содержат данные одного типа.

Атрибут, значение которого однозначно идентифицирует кортежи, называется ключевым (или просто ключом). В нашем случае ключом является атрибут «Табельный номер», поскольку его значение уникально для каждого работника предприятия. Если кортежи идентифицируются только сцеплением значений нескольких атрибутов, то говорят, что отношение имеет составной ключ.

Отношение может содержать несколько ключей. Всегда один из ключей объявляется первичным, его значения не могут обновляться. Все остальные ключи отношения называются возможными ключами.

В отличие от иерархической и сетевой моделей данных в реляционной отсутствует понятие группового отношения. Для отражения ассоциаций между кортежами разных отношений используется дублирование их ключей. Пример базы данных, содержащей сведения о подразделениях предприятия и работающих в них сотрудниках, применительно к реляционной модели будет иметь следующий вид (рис. 22).

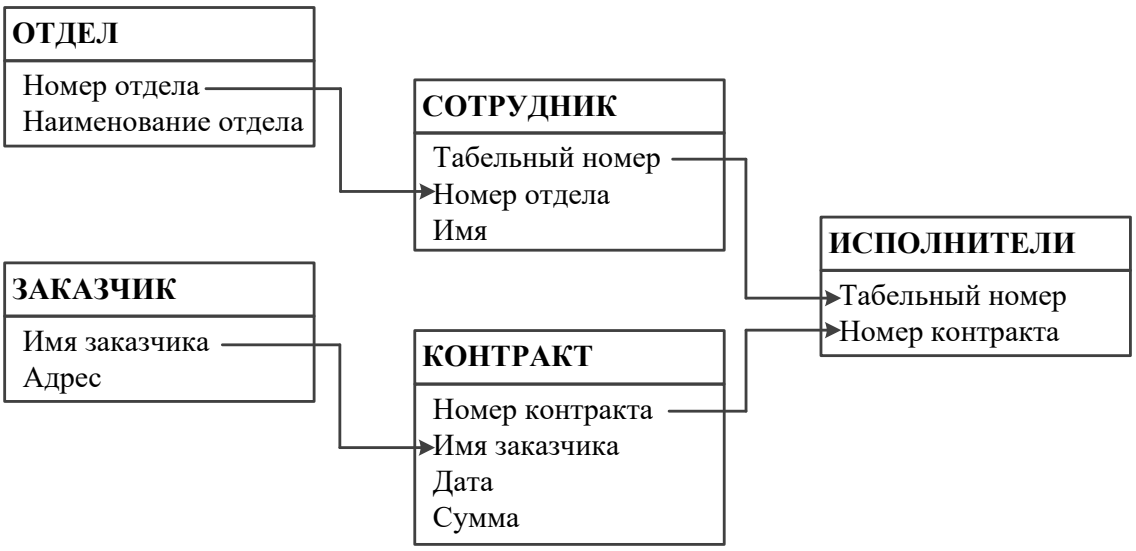


Рис. 22. База данных о подразделениях и сотрудниках предприятия

В состав теоретико-множественных операций входят операции:

– объединения отношений;
– пересечения отношений;
– взятия разности отношений;
– декартова (прямого) произведения отношений.

Специальные реляционные операции включают:

– ограничение отношения (селекция, выборка);
– проекцию отношения;
– соединение отношений
– деление отношений.

Кроме того, в состав реляционной алгебры включается операция присваивания, позволяющая сохранить в базе данных

результаты вычисления алгебраических выражений, и операция переименования атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

Операндами в операциях реляционной алгебры и результатами операций являются отношения. В операциях объединения, пересечения, взятия разности, прямого произведения, соединения и деления участвуют два операнда, в остальных — один.

Результатом объединения (пересечения) отношений является отношение, включающее кортежи, которые входят в оба (хотя бы в одно) из отношений-операндов.

Объединение отношений R и S, обозначаемое как $R \cup S$, представляет собой множество кортежей, которые принадлежат R или S либо им обоим. Оператор объединения применяется только к отношениям одной и той же арности. Поэтому все кортежи в результате имеют одинаковое число компонентов, кортеж d,a,f принадлежит обоим отношениям одновременно (табл. 3).

Таблица 3

Результат операции объединение отношений

a_R	b_R	c_R
d	a	f
c_R	b_R	d_R
b_S	g_S	a_S

Результатом разности отношений будет отношение, включающее кортежи, которые входят в отношение — первый операнд и не входят в отношение — второй операнд.

Разностью отношения R и S, обозначаемой как $R-S$, называется множество кортежей, принадлежащих R, но не принадлежащих S. Здесь снова требуется, чтобы R и S имели одну и ту же арность. В табл. 4 показан результат этой операции.

Таблица 4

Результат операции разности отношений

a_R	b_R	c_R
c_R	b_R	d_R

Результат прямого произведения отношений — отношение, кортежи которого являются конкатенацией (сцеплением) кортежей первого и второго операндов.

Пусть R и S — отношения арности $k1$ и $k2$ соответственно. Тогда декартовым произведением отношений R и S называется множество кортежей длины $k1 + k2$, первые $k1$ компонентов которых образуют кортежи, принадлежащие R, а последние $k2$ — кортежи, принадлежащие S. В табл. 5 приведен указанный результат.

Таблица 5

Результат операции прямого произведения отношений

A	B	C	D	E	F
a_R	b_R	c_R	b_S	g_S	a_S
a_R	b_R	c_R	d_S	a_S	f_S
d_R	a_R	f_R	b_S	g_S	a_S
d_R	a_R	f_R	d_S	a_S	f_S
c_R	b_R	d_R	b_S	g_S	a_S
c_R	b_R	f_R	d_S	a_S	f_S

Результатом ограничения (селекции) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющие этому условию.

Пусть F — формула, образованная следующими средствами:

а) операндами, являющимися константами или номерами компонентов;

б) арифметическими операторами сравнения;

в) логическими операторами (И), (ИЛИ), (НЕ).

В этом случае $\sigma f(R)$ есть множество кортежей t, принадлежащих R, таких, что при подстановке i-го компонента t вместо любого вхождения номера i в формулу F для всех i она станет истинной. Например, $\sigma 2 > 3(R)$ обозначает множество кортежей, принадлежащих R, второй компонент которых больше третьего компонента. Или селекция $\sigma 2 > 3(R)$ определяет множество кортежей, принадлежащих R, второй компонент (B) которых есть bR. Отношение $\sigma B > bR(R)$ приведено в табл. 6.

Таблица 6

Результат операции ограничения (селекции) отношения

A	*B*	*C*
A	*B*	*c*
C	*B*	*d*

Результатом проекции отношения на заданный набор его атрибутов будет отношение, кортежи которого получаются путем взятия соответствующих значений из заданных столбцов кортежей отношения-операнда. Если при этом возникают кортежи-дубликаты, они уничтожаются.

Суть этой операции заключается в том, что берется отношение *R*, удаляются некоторые из его компонентов и (или) переупорядочиваются оставшиеся компоненты. Пусть *R* — отношение арности *k*. Обозначим через $i_1, i_2, \ldots, i_m(R)$, где *ij* являются различными целыми числами в диапазоне от 1 до *k*, проекцию на компоненты $i_1, i_2, \ldots, i_m$, т. е. множество компонентов $\{a_1, a_2, \ldots, a_m\}$, таких, что существует некоторый принадлежащий *R* кортеж $(b_1, b_2, \ldots, b_k)$ длины *k*, удовлетворяющий условию $a_j = b_{ij}$ для $j = 1, 2, ..., m$. Например, для построения нужно из каждого кортежа *t*, принадлежащего *R*, сформировать кортеж длины 2 из третьего и первого его компонентов в указанном порядке. Пример проекции приведен в табл. 7.

Таблица 7

Результат операции проекции отношения

C	*A*
c_R	a_R
f_R	d_R
d_R	c_R

2.4. Постреляционная модель

Постреляционная модель данных представляет собой расширенную реляционную модель, в которой отменено требование атомарности атрибутов. Поэтому постреляционную модель называют «не первой нормальной формой» (NF2) или «многомерной

базой данных». Она использует трехмерные структуры, позволяя хранить в полях таблицы другие таблицы. Тем самым расширяются возможности по описанию сложных объектов реального мира. В качестве языка запросов используется несколько расширенный SQL, позволяющий извлекать сложные объекты из одной таблицы без операций соединения [13].

Существует несколько коммерческих постреляционных СУБД, более подробные сведения о них можно получить на вебсерверах фирм-производителей. Пожалуй, самыми известными из них являются системы Adabas, Pick и Universe [13].

Важным аспектом традиционной реляционной модели данных является тот факт, что элементы данных, которые хранятся на пересечении строк и столбцов таблицы, должны быть неделимы и единственны. Это значит, что данные не могут быть развернуты в процессе дальнейшей обработки. Такое правило было заложено в основу реляционной алгебры при ее разработке как математической модели данных. Дальнейшие исследования показали, что существует ряд случаев, когда ограничения классической реляционной модели существенно мешают эффективной реализации приложений [12, с. 73].

Опыт разработки прикладных информационных систем показал, что отказ от этой установки ведет к качественно полезному расширению модели данных. Если допустить, что значение данных может само состоять из подзначений, то в результате возникает понятие многозначного поля. Проще всего рассматривать набор многозначных полей в таблице как самостоятельную встроенную таблицу.

При условии, что встроенная таблица удовлетворяет общим критериям (например, имеет уникальный ключ), естественным образом происходит расширение операторов реляционной алгебры.

Некоторые постреляционные модели данных могут поддерживать также ассоциированные многозначные поля, которые часто называют множественными группами. Иными словами, вы можете связать несколько столбцов с множественными значениями в единое целое, называемое ассоциацией. При этом в строке первое значение одного столбца ассоциации соответствует пер-

вым значениям всех других столбцов ассоциации, в такой же связи находятся все вторые значения столбцов и т. д. [12, с. 72].

О таблицах, содержащих многозначные поля, говорят, что они находятся в не первой нормальной форме, или NF2. Как было замечено ранее, при условии, что используемые поля подчиняются определенным правилам, позволяющим обращаться с ними, как с таблицами, встроенными в другие таблицы, форма NF2 не нарушает принципы реляционной алгебры. Более того, такая информация полностью доступна, так как расширенные операторы, которые работают с таблицами NF2, позволяют извлекать встроенные таблицы и рассматривать данные как информацию, поступившую из таблиц 1NF [12, с. 74].

Подводя итог вышесказанному, можно дать определение второй нормальной формы: отношение находится в NF2, если оно находится в 1NF и каждый неключевой атрибут функционально полно зависит от ключа.

Далее рассмотрим основные направления исследований и разработок в области постреляционных систем [12, с. 75−79].

1. Базы сложно-структурированных объектов (реляционная модель с отказом от первой нормальной формы).

2. Активные базы данных. По определению БД называется активной, если СУБД по отношению к ней выполняет не только те действия, которые явно указывает пользователь, но и дополнительные действия в соответствии с правилами, заложенными в саму БД. Основа этой идеи содержалась в языке SQL. На самом деле триггер — это введенное в БД правило, в соответствии с которым СУБД должна производить дополнительные действия. Однако триггеры не были полностью реализованы ни в одной из известных систем.

3. Дедуктивные базы данных. По определению, дедуктивная БД состоит из двух частей: экстенсиональной, содержащей факты, и интенсиональной, содержащей правила для логического вывода новых фактов на основе экстенсиональной части и запроса пользователя. При таком общем определении SQL-ориентированную реляционную СУБД можно отнести к дедуктивным системам. Действительно, определенные в схеме реляционной БД представления — интенсиональная часть БД. Ос-

новным отличием реальной дедуктивной СУБД от реляционной является то, что и правила интенсиональной части БД, и запросы пользователей могут содержать рекурсию.

4. Темпоральные базы данных. Реляционная модель не поддерживает время как особую переменную, хотя и располагает в SQL некоторыми функциями обработки данных типа date и time. Обычные БД хранят мгновенный снимок модели предметной области. Любое изменение в момент времени *t* некоторого объекта приводит к недоступности состояния этого объекта в предыдущий момент времени. В большинстве развитых СУБД предыдущее состояние объекта сохраняется в журнале изменений, но возможности доступа со стороны пользователя нет. Данная проблема часто решается путем явного ввода в хранимые отношения явного временного атрибута и поддержнием его значения на уровне приложений. Так в стандарте SQL появились специальные типы данных date и time.

5. Интегрированные, или федеративные, системы и мультибазы данных. Направление интегрированных, или федеративных, систем неоднородных БД и мульти-БД появилось в связи с необходимостью комплексирования систем БД, основанных на разных моделях данных и управляемых разными СУБД. Основной задачей интеграции неоднородных БД является предоставление пользователям интегрированной системы глобальной схемы БД, представленной в некоторой модели данных, и автоматическое преобразование операторов манипулирования БД глобального уровня в операторы, понятные соответствующим локальным СУБД. В теоретическом плане проблемы преобразования решены.

6. «СУБД следующего поколения». Термин «системы следующего (или третьего) поколения» вошел в жизнь после опубликования группой известных специалистов в области БД «Манифеста систем баз данных третьего поколения». Сторонники этого направления придерживаются принципа эволюционного развития возможностей СУБД без коренного изменения предыдущих подходов и с сохранением совместимости с системами предыдущего поколения. В целом СУБД следующего поколения — это прямые наследники реляционных систем.

Частично требования к системам следующего поколения означают просто необходимость реализации или усиления давно известных свойств, отсутствующих в большинстве текущих реляционных СУБД (ограничения целостности, триггеры, модификация БД через VIEW и т. д.). В число новых требований входят:

– полнота системы типов, поддерживаемых в СУБД;

– поддержка иерархии и наследования типов;

– возможность управления сложными объектами и т. д.

7. Многомерные модели. Многомерные модели рассматривают данные либо как факты с соответствующими численными параметрами, либо как текстовые измерения, которые характеризуют эти факты. Источником для многомерных баз данных послужили не современные технологии баз данных, а алгебра многомерных матриц, которая использовалась для ручного анализа данных с конца XIX столетия.

Электронные таблицы не подходят для управления и хранения многомерных данных, поскольку они слишком жестко связывают данные с их внешним видом, не отделяя структурную информацию от желаемого представления информации. Скажем, добавление третьего измерения, такого как время, или группировка данных по обобщенным типам продуктов требует значительно более сложной настройки. Многомерные базы данных рассматривают данные как кубы, которые являются обобщением электронных таблиц на любое число измерений. Кроме того, кубы поддерживают иерархию измерений и формул без дублирования их определений. Набор соответствующих кубов составляет многомерную базу данных (хранилище данных).

Кубами легко управлять, добавляя новые значения измерений. В обычном обиходе этим термином обозначают фигуру с тремя измерениями, однако теоретически куб может иметь любое число измерений. На практике чаще всего кубы данных имеют от 4 до 12 измерений. Современный инструментарий часто сталкивается с нехваткой производительности, когда так называемый гиперкуб имеет свыше 10−15 измерений.

Это порождает несколько важных тенденций. Данные, которые необходимо анализировать, становятся все более распределенными. К примеру, это часто необходимо для выполнения анализа, при котором используются данные в формате XML,

получаемые с определенных веб-сайтов. Растущая распределенность данных, в свою очередь, требует применения методов, которые позволяют легко добавлять новые данные в многомерные базы данных, тем самым упрощая задачу создания интегрированного хранилища данных. Среди примеров — автоматическая генерация измерений и кубов из новых источников данных и методы простой и динамической очистки данных.

2.5. Объектно-ориентированная модель

Направление объектно-ориентированных баз данных (ООБД) возникло сравнительно давно. Публикации появлялись уже в середине 1980-х гг. Однако наиболее активно это направление развивается в последние годы. С каждым годом увеличивается число публикаций и реализованных коммерческих и экспериментальных систем.

Возникновение направления ООБД определяется необходимостью разработки сложных информационных прикладных систем, для которых технология предшествующих систем БД не была вполне удовлетворительной.

Основная практическая функция ООБД связана с потребностью в некоторой интегрированной среде построения сложных информационных систем. В этой среде должны отсутствовать противоречия между структурной и поведенческой частями проекта и должно поддерживаться эффективное управление сложными структурами данных во внешней памяти. С этой точки зрения языковая среда ООБД — объектно-ориентированная система программирования, естественно включающая средства работы с долговременными объектами [12, с. 88].

Объектно-ориентированная модель данных включает три аспекта:

– структурный;

– целостный;

– манипуляционный.

Структура объектной модели описывается с помощью трех ключевых понятий [12, с. 89–91].

1. Инкапсуляция — каждый объект обладает некоторым внутренним состоянием (хранит внутри себя запись данных), а также набором методов — процедур, с помощью которых (и только таким образом) можно получить доступ к данным, определяющим внутреннее состояние объекта, или изменить их. Таким образом, объекты можно рассматривать как самостоятельные сущности, отделенные от внешнего мира. Пример:

```
Class Point {       // вводим новый тип данных - объект "точка"
X,Y : int;          // данные объекта - координаты точки
.........
Point(X : int, Y : int);  // конструктор объекта - процедура, вызываемая
                          // при определении переменной на базе объекта и
                          // присваивающая значения его данным
.........
Draw();                 // метод "нарисовать точку"
Erase();                // метод "стереть точку"
Move(newX,newY);        // метод "переместить точку" (изменяет данные
                           // объекта)
int GetX();             // метод "получить значение поля X"
int GetY();             // метод "получить значение поля Y"
.........
              // все методы должны быть описаны, например
              // реализация метода Move:

Move(newX : int, newY : int) {
X=newX;                 // запись новых данных в объект
Y=newY;                 //
}
}                       // конец описания объекта

Begin                   // основная процедура программы

Point A(0,0);       // создать новый объект и присвоить ему данные

for i=1 to 100          // создать цикл
A.Draw();             // нарисовать точку
A.Hide();             // стереть точку
A.Move(i,i*10);     // присвоить экземпляру объекта новые данные
endfor;                 //
print(A.GetX(),A.GetY());     // получить и напечатать данные объекта
End.
```

Из этого примера видно, что мы не можем напрямую обратиться к данным объекта, а должны вызывать метод Move для изменения его данных и GetX, GetY для считывания значений этих данных. Объект скрывает свою внутреннюю структуру, именно это свойство и называется «инкапсуляцией».

2. Наследование — возможность создавать из классов объектов новые классы объектов, которые наследуют структуру и методы своих предков, добавляя к ним черты, отражающие их собственную индивидуальность. Наследование может быть простым (один предок) и множественным (несколько предков). Пример:

```
Class Circle extend Point {       // создаем новый объект "окружность",
                                  // наследующий свойства объекта "точка"
Radius : int;  // добавляем новое поле "радиус", поля X и Y наследуются
               // от родительского объекта
.............
Circle(X:int,Y:int,Radius:int);    // конструктор нового объекта
.............
Draw();            // переопределяем некоторые методы
Hide();            // родительского объекта, метод Move наследуется
.............
ChangeRadius(Radius); // вводим новый метод "изменить радиус"
.............
GetRadius();           // вводим новый метод "получить значение радиуса"
          // методы GetX и GetY наследуются от родительского объекта
          }
```

3. Полиморфизм — различные объекты могут по-разному реагировать на одинаковые внешние события в зависимости от того, как реализованы их методы. Пример:

```
Begin
Point A(100,100);
Circle B(200,200,50);
A.Draw();       // рисует точку
B.Draw();       // рисует окружность
End.
```

Основные трудности объектно-ориентированного моделирования данных проистекают из того, что такого развитого

математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных, не существует. Поэтому до сих пор нет базовой объектно-ориентированной модели.

Классическим представителем объектно-ориентированного подхода является СУБД O2. В O2 поддерживаются объекты и значения.

Объект — это пара (идентификатор, значение), причем объекты инкапсулированы, т. е. их значения доступны только через методы — процедуры, привязанные к объектам.

Значения могут быть атомарными или структурными.

Структурные значения строятся из значений или объектов, представленных своими идентификаторами, с помощью конструкторов множеств, кортежей и списков.

Элементы структурных значений доступны с помощью предопределенных операций (примитивов). Возможны два вида организации данных:

– классы, экземплярами которых являются объекты, инкапсулирующие данные и поведение,

– типы, экземплярами которых являются значения.

Каждому классу сопоставляется тип, описывающий структуру экземпляров класса. Типы определяются рекурсивно на основе атомарных типов и ранее определенных типов и классов с применением конструкторов. Поведенческая сторона класса определяется набором методов.

Объекты и значения могут быть именованными. С именованием объекта или значения связана долговременность его хранения:

– любые именованные объекты или значения долговременны;

– любые объект или значение, входящие как часть в другой именованный объект или значение, долговременны.

С помощью специального указания, задаваемого при определении класса, можно добиться долговременности хранения любого объекта этого класса. В этом случае система автоматически порождает значение-множество, имя которого совпадает с именем класса. В этом множестве гарантированно содержатся все объекты данного класса.

Метод — это программный код, привязанный к конкретному классу и применимый к объектам этого класса. Определение метода в O2 производится в два этапа. Сначала объявляется сигнатура метода, т. е. его имя, класс, типы или классы аргументов и тип или класс результата. Методы могут быть публичными (доступными из объектов других классов) или приватными (доступными только внутри данного класса).

Потребность поддержания в объектно-ориентированной СУБД не только языка (семейства языков) программирования ООБД, но и развитого языка запросов в настоящее время признается практически всеми разработчиками.

В рамках данной потребности существует три подхода к формированию языка:

– первый подход — языки, являющиеся объектно-ориентированными расширениями языков запросов реляционных систем. Наиболее распространены языки с синтаксисом близким к известному языку SQL. Это связано с его общим признанием и чрезвычайно широким распространением;

– второй подход основывается на построении полного логического объектно-ориентированного исчисления. По поводу построения такого исчисления имеются теоретические работы, но законченный и практически реализованный язык запросов неизвестен;

– третий подход основывается на применении дедуктивного подхода. В основном это отражает стремление разработчиков к сближению направлений дедуктивных и объектно-ориентированных БД.

Стандарт ODMG. ODMG (Object Data Management Group) — консорциум поставщиков ООБД и других заинтересованных организаций, созданный в 1991 г. Его задачей является разработка стандарта на хранение объектов в базах данных.

ODMG добавляет возможности взаимодействия с базами данных в объектно-ориентированные языки программирования: определяются средства долговременного хранения объектов и расширяется семантика языка, вносятся операторы управления данными. Стандарт состоит из нескольких частей.

1. Объектная модель — унифицированная основа всего стандарта. Она расширяет объектную модель консорциума ODMG

за счет введения таких свойств, как связи и транзакции для обеспечения функциональности, требуемой при взаимодействии с базами данных. Ключевые концепции объектной модели ODMG:

– наделение объектов такими свойствами, как атрибуты и связи;

– методы объектов (поведение);

– множественное наследование;

– идентификаторы объектов (ключи);

– определение таких совокупностей объектов, как списки, наборы, массивы и т. д.;

– блокировка объектов и изоляция доступа;

– операции над базой данных.

2. Язык описания объектов (ODL — Object Defifnition Language) — средство определения схемы базы данных (по аналогии с DDL в реляционных СУБД). ODL создает слой абстрактных описаний так, что схема базы данных становится независима как от языка программирования, так и от СУБД.

3. Язык объектных запросов (OQL — Object Query Language) — SQL-подобный декларативный язык, который предоставляет эффективные средства для извлечения объектов из базы данных, включая высокоуровневые примитивы для наборов объектов и объектных структур.

Из числа архитектур с традиционной организацией наибольшее влияние на объектно-ориентированные СУБД оказали реляционные системы. Многие объектно-ориентированные системы строятся над некоторой существующей реляционной СУБД. Кроме такого применения реляционных систем, для упрощения разработки объектно-ориентированной СУБД развитые в реляционных СУБД методы применяются и в заново разрабатываемых объектно-ориентированных системах.

Непосредственным предшественником объектно-ориентированных СУБД являются системы, поддерживающие организацию сложных объектов (постреляционные системы). Эти системы большей частью появились по причине несоответствия возможностей реляционных СУБД потребностям нетрадиционных приложений (автоматизация проектирования, инженерия и т. д.). В таких системах частично поддерживается структурная часть ООБД (без возможностей наследования).

Другой основой объектно-ориентированных СУБД являются расширяемые системы. Основная идея таких систем состоит в поддержании набора модулей с четко оговоренными интерфейсами, на базе которого можно быстро построить СУБД, опирающуюся на конкретную модель данных или предназначенную для конкретной области применений.

2.6. XML-данные

Расширяемый язык разметки (Extensible Markup Language, XML) описывает класс объектов XML-document, а также частично описывает работу компьютерных программ, обрабатывающих объекты с данными, реализующими этот класс. XML — это прикладной уровень или усеченная форма SGML — стандартного обобщенного языка разметки. По своему построению XML-документ является полноценным SGML документом.

Язык XML был разработан группой XML Working Group (первоначально называемой SGML Editorial Review Board), сформированной в 1996 г. под патронатом World Wide Web Consortium (W3C).

XML-документ. Объект данных становится XML документом, если в соответствии с определениями обсуждаемой спецификации он является корректным. Корректный XML-документ также может стать действительным, если отвечает некоторым дополнительным ограничениям.

Каждый XML имеет логическую и физическую структуру. Физически документ состоит из элементов, называемых сущностями. Любая сущность может ссылаться на другие сущности, обеспечивая их включение в данный документ. Документ начинается с «корня» или сущности документа. С логической точки зрения, документ строится из деклараций, элементов, комментариев, ссылок на символ и инструкций обработки. Все они размечаются в документе явным образом.

Корректный XML-документ. Текстовый объект становится корректным (well-formed) XML-документом, если:

1) как единое целое он соответствует сценарию document;

2) отвечает всем ограничениям корректности, представленным в этой спецификации;

3) все разобранные сущности, на которые в данном документе прямо или косвенно делается ссылка, являются корректными.

Преобразование в языке XSLT предстает в виде корректного XML-документа, соответствующего требованиям для пространств имен из Рекомендации XML. Оно может содержать как элементы, определяемые в XSLT, так и элементы, которые в XSLT не определены. Элементы, определяемые в XSLT, отличаются принадлежностью к конкретному пространству имен XML, которое в данной спецификации называется пространством имен XSLT.

Преобразование, выраженное через XSLT, описывает правила преобразования исходного дерева документа в конечное дерево. Преобразование строится путем сопоставления образцов и шаблонов. Образец сравнивается с элементами исходного дерева, а шаблон используется для создания частей конечного дерева. Конечное дерево отделено от исходного дерева. Структура конечного дерева может полностью отличаться от структуры исходного дерева. В ходе построения конечного дерева элементы исходного дерева могут подвергаться фильтрации и переупорядочению, также может быть добавлена новая структура.

Преобразование, выраженное через XSLT, называется стилем (stylesheet). Это сделано для того, чтобы в случае, когда XSLT приводится к словарю форматирования XSL, данное преобразование выполняло функции стиля.

Не конкретизируется, каким образом стиль XSLT привязывается к документу XML. Рекомендуется, чтобы процессоры XSL поддерживали механизм, описанный в документе. Когда этот или какой-либо другой механизм дает последовательность из нескольких стилей XSLT, которые должны быть применены к XML-документу одновременно, это равносильно ситуации, как если бы использовался один стиль, поочередно импортирующий все члены этой последовательности стилей.

Стиль содержит набор правил шаблона. Правило шаблона состоит из двух частей: образца, который сопоставляется с узлами в исходном дереве, и шаблона, который может быть обработан для формирования фрагмента в конечном дереве. Такая схема позволяет использовать один стиль для большого класса документов, имеющих одинаковую структуру исходного дерева.

По соглашению, названия атрибутов и функций для всех элементов XSLT пишутся прописными буквами, для разделения слов используют дефис, а сокращения используются только в том случае, если они уже есть в синтаксисе соответствующего языка, такого как XML или HTML.

Выражения в XSLT используются для таких целей, как:

– выбор узлов для обработки;

– формулирование условий для разных вариантов обработки узла;

– генерация текста для подстановки в конечное дерево.

Выражение должно соответствовать сценарию Expr из XPath.

Выражения используются в значении определенных атрибутов у элементов, описанных в XSLT, а также — в фигурных скобках — в шаблоне для значения атрибута.

Внешнее выражение в XSLT (выражение, не являющееся частью какого-либо другого выражения) получает свой контекст следующим образом:

– узел контекста получается из текущего узла;

– положение в контексте определяется положением текущего узла в текущем наборе узлов, первая позиция имеет индекс 1;

– размер контекста определяется размером текущего набора узлов;

– привязка переменных контекста — это привязка в пределах видимости того элемента, которому принадлежит атрибут с рассматриваемым выражением;

– набор деклараций пространства имен — это те декларации, в области видимости которых находится элемент, содержащий атрибут с рассматриваемым выражением. Сюда относится также неявная декларация префикса xml, обязательная согласно W3C Namespaces Recommendation. Пространство имен по умолчанию к этому набору деклараций не относится;

– используемая библиотека функций формируется из основной библиотеки функций, дополнительных функций, а также функций расширения. Если в выражении появится вызов какой-либо другой функции, зафиксируется ошибка.

Реализация XMLT может обрабатывать исходный документ любым способом, который дает тот же результат, как если бы выполнялась обработка в соответствии с представленной моделью.

Язык XPath является результатом попыток создать единые синтаксис и семантику для функционала, совместно используемого XSL Transformations и XPointer.

Главная задача языка XPath — адресация частей в XML-документе. Для достижения этой цели язык дополнительно наделен основными функциями для манипулирования строками, числами и булевыми значениями. В XPath используется компактный синтаксис, отличный от принятого в XML, облегчающий использование языка XPath при записи адресов URI и значений атрибутов XML. XPath работает не с внешним синтаксисом XML-документа, а с его абстрактной логической структурой. XPath получил такое название потому, что использовался в URL для записи путей, обеспечивающих навигацию по иерархической структуре XML документа.

Язык XPath спроектирован так, что помимо поддержки адресации он обладает естественным набором элементов, которые могут использоваться для сравнения (проверки, соответствует ли узел некому шаблону). Такой порядок использования языка XPath описывается в спецификации XSLT.

XPath представляет XML-документ в виде дерева узлов. Узлы бывают различных типов, например, узлы элементов, узлы атрибутов и узлы текста. Для каждого типа узлов в XPath определяется способ вычисления строкового значения. Некоторые типы узлов имеют также имя. XPath полностью поддерживает пространство имен XML. В результате имя любого узла в этом языке образуется из двух частей: локальной и URI некого пространства имен (возможно, нулевого), такая комбинация называется расширенным именем.

Главной синтаксической конструкцией языка XPath является выражение. Любое выражение соответствует сценарию Expr. В результате обработки выражения получается объект, относящийся к одному из четырех основных типов:

– набор узлов (node-set) — неупорядоченный набор узлов без дубликатов;

– булево значение (boolean) — true или false;

– число (number) — число с плавающей точкой;

– строка (string) — последовательность UCS символов.

Обработку выражений осуществляют, отталкиваясь от некого контекста. В спецификациях XSLT и XPointer указывается, каким образом в них определяется контекст для выражений XPath. Контекст образуется из:

– узла (узел контекста, context node);

– пары ненулевых положительных целых чисел (положение в контексте и размер контекста);

– привязки переменных контекста (variable bindings);

– библиотеки функций;

– набора деклараций пространства имен в области видимости данного выражения.

Выражения XPath часто используются в атрибутах XML. Строковые значения, используемые в выражениях, заключаются в одинарные или двойные кавычки, которые используются также для записи атрибутов XML. Поэтому, чтобы символ кавычки в этом выражении не интерпретировался XML процессором как конец значения атрибута, его необходимо записывать как ссылку на символ (" или '). Впрочем, если атрибут XML был заключен в двойные кавычки, в выражении можно свободно использовать символы одинарных кавычек, и наоборот.

Другим важным типом выражений является путь адресации (location path). Путь адресации выбирает некое множество узлов, отталкиваясь от некоторого узла контекста. Ниже приводятся примеры путей адресации, использующих развернутый синтаксис:

child::para находит элемент para, являющийся непосредственным потомком узла контекста;

child::* собирает все элементы, являющиеся непосредственными потомками узла контекста;

child::text() собирает все текстовые узлы, являющиеся непосредственными потомками узла контекста;

child::node() собирает все непосредственные потомки текущего узла контекста независимо от типа этих узлов;

attribute::name в текущем узле контекста находит атрибут name;

attribute::* собирает все атрибуты в текущем узле контекста;

descendant::para среди потомков узла контекста находит элементы para;

ancestor::div собирает все предки div текущего узла контекста;

ancestor-or-self::div собирает предки div текущего узла контекста и также, если сам узел контекста тоже является элементом div, включает в набор и его;

descendant-or-self::para среди потомков узла контекста выбирает элементы para, а также, если сам узел контекста является элементом para, то включает в набор и его.

Пути адресации бывают двух типов: относительные и абсолютные.

Относительный путь адресации состоит из последовательности одного или нескольких шагов адресации, отделенных друг от друга символом /. Шаги в относительном пути адресации считаются слева направо. На каждом шаге осуществляется отбор узлов, отталкиваясь от некоторого узла контекста. Основная последовательность шагов образуется последовательным перечислением шагов в порядке их выполнения. Сперва, отталкиваясь от узла контекста, последовательность шагов набирает некий набор узлов. Затем каждый узел в этом наборе поочередно используется как узел контекста для следующего шага. Все наборы узлов, полученных после выполнения такого шага, опять собираются вместе. В итоге в полученном объединении будет собран набор узлов, идентифицируемых данной последовательностью шагов. Например, формула child::div/child::para собирает все элементы para, являющиеся непосредственными потомками элемента div, который сам является непосредственным потомком узла контекста, или, иными словами, находит все элементы — потомки во втором поколении para, родителями которых являются div.

Абсолютный путь адресации состоит из символа /, за которым может следовать относительный путь адресации. Сам символ / находит корневой узел документа, в котором располагался текущий узел контекста. Если за этим символом последовал относительный путь адресации, то получившийся путь адре-

сации будет собирать набор узлов так, как если бы в качестве узла контекста был выбран корневой узел того документа, которому принадлежал действительный текущий узел контекста.

XQuery — это мощный и удобный язык, предназначенный для обработки XML. Под XML понимаются не только файлы в XML-формате, но и другие данные, похожие на XML, включая базы данных, структура которых представляет собой вложенные, поименованные деревья с атрибутами.

В XQuery любая конструкция — это выражение, результатом вычисления которого является некоторое значение. Программа XQuery или скрипт — это просто выражение вместе с некоторыми необязательными функциями и другими определениями.

Примитивные типы данных XQuery такие же, как в XML Schema:

– числа, включая целые и числа с плавающей запятой;

– булевы числа — true (истина) и false (ложь);

– строки символов, например, "Hello world!". Строки являются неизменными (immutable), т. е. символ в строке не может быть изменен;

– различные типы для представления дат, времени и продолжительности;

– несколько типов, связанных с XML. Например, QName — это часть локального имени (как template) и URL (унифицированный указатель ресурса), который используется для представления имени тега после того, как для него было установлено пространство имен (как xsl:template).

Производные типы являются вариациями или ограничениями других типов. Примитивные типы и типы, полученные из них, известны как атомарные типы, поскольку атомарные величины не содержат других величин.

В XQuery также есть типы данных необходимые для представления величин XML. Для этого используются величины узлов следующих семи типов:

– элемент;

– атрибут;

– пространство имен;

– текст;

– комментарий;
– инструкция обработки;
– документ (корень).

В некоторых реализациях XQuery используются объекты DOM для задания величин узлов, хотя в реализациях могут применяться и другие представления.

Для создания и возврата узлов используются различные стандартные функции XQuery. Так, функция document читает XML-файл, указанный аргументом URL, и возвращает корневой узел документа.

Распространенные процессоры шаблонов, такие как JSP, ASP и PHP, позволяют вставлять выражения на языке программирования в содержимое HTML. Помимо этого XQuery разрешает помещать формы XML/HTML внутрь выражений и использовать их в качестве значений переменных и параметров.

Величины узлов XQuery являются неизменными (после создания узла его нельзя изменить).

Рассмотренные атомарные величины (числа, строки и т. п.) и величины узлов (элементы, атрибуты и т. д.) известны как простые величины. Результатом вычисления выражения XQuery на самом деле является последовательность простых величин. Например, «3, 4, 5» — это последовательность, состоящая из трех целых. Если последовательность содержит только одно значение, то она совпадает с самой величиной. Не допускается использовать вложения для последовательностей.

В XQuery используются выражения XPath. XQuery можно рассматривать как обобщение XPath. За исключением некоторых малоизвестных форм (в основном необычных «осевых спецификаторов») все выражения XPath являются также и выражениями XQuery.

Необходимо отметить следующее различие между XPath и XQuery: выражения XPath могут вернуть множество узлов (node set), а такое же выражение XQuery возвращает последовательность узлов. Для обеспечения совместимости эти последовательности находятся в документальном порядке, в них удалены дубликаты — благодаря этому они эквиваленты множествам.

XSLT очень удобен для выражения довольно простых преобразований, но более сложные таблицы стилей (особенно

что-либо с нетривиальной логикой или программированием) часто можно записать более аккуратно с помощью XQuery.

XQuery — строго типизированный язык программирования. Как Java и C#, XQuery — это смесь статического (проверка совместимости типов во время компиляции) и динамического контроля типов (тестирование типов во время выполнения). Однако типы в XQuery отличаются от классов, присущих объектно-ориентированному программированию. Взамен XQuery включает типы, которые соответствуют модели данных XQuery, и позволяет импортировать типы из XML Schema.

2.7. Причины появления нереляционных СУБД

Основным принципом цифровой трансформации является переход от централизованной к распределенной обработке данных, что выражается в появлении ряда новых технологий.

Этот принцип реализуется в настоящее время практически повсеместно. В рамках цифровой экономики одной из наиболее успешных реализаций принципа децентрализации хранения и обработки данных является технология blockchain, в том числе ее наиболее удачное применение для создания криптовалют (рис. 23).

Применительно к обработке данных промышленных предприятий реализация принципа децентрализации обусловлена появлением настолько большого количества данных, что традиционные (централизованные) методы их обработки оказались неэффективными, а зачастую и просто неприменимыми. Становится невозможным хранение и обработка всей информации промышленного предприятия силами только одного узла, и разумным решением в данном случае является задействование вычислительных ресурсов нескольких узлов с размещением на них данных на основе принципа «разделения» (размещения части данных предприятия) или «репликации» (дублирования данных на всех узлах). С этим связан и переход от реляционных СУБД к нереляционным (NoSQL), имеющим распределенную природу и, как следствие, более простую и гибкую модель данных. В них отсутствуют жесткие структуры данных, они основаны либо на использовании единственного уникального ключа к строкам

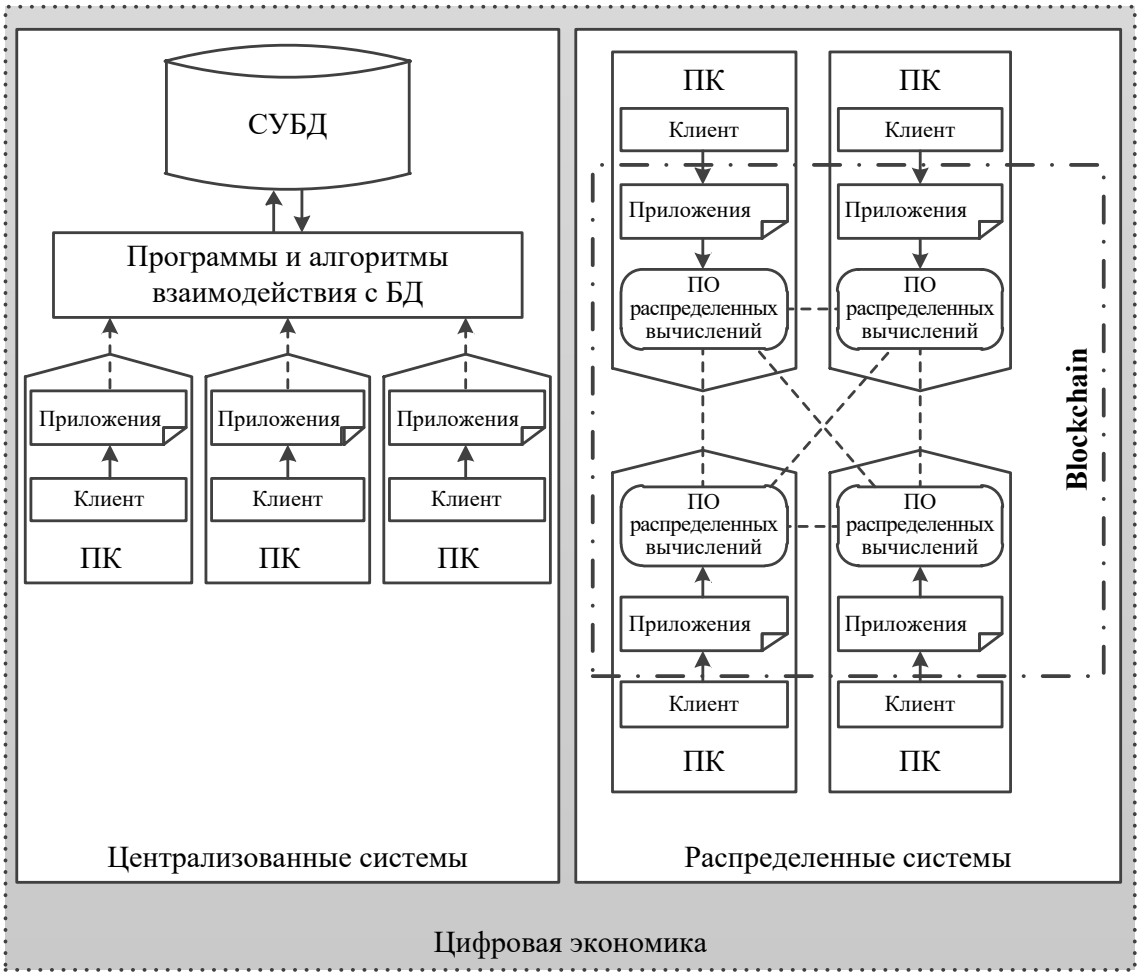


Рис. 23. Роль технологии Blockchain в цифровой экономике

данных (системы «ключ-значение»), либо на хранении отдельных документов произвольной структуры и длины («документные» системы), либо на принципе инвертирования (системы типа «Google BigTable»).

Поскольку никакой сколь угодно мощный сервер при решении современных задач в принципе не способен в одиночку обеспечить нужную производительность, основными принципами NoSQL стали отказ от реляционной модели для учета специфики обрабатываемых данных, а также хорошая горизонтальная масштабируемость до сотен и тысяч серверов для обеспечения скорости работы. Однако это выявило еще одну проблему, сформулированную в виде теоремы CAP (Consistence, Availability, Partition tolerance — «согласованность, доступность, устойчивость к разделению»): в распределенной вычислительной системе невозможно одновременно выполнить требования

по согласованности данных, доступности системы и устойчивости к разделению. Под последним требованием понимается то, что система не распадается на несколько изолированных секций, внутри которых выполняются требования по согласованности и доступности [14].

Нестрогое доказательство теоремы CAP основано на простых рассуждениях. Пусть распределенная система состоит из N серверов, каждый из которых обрабатывает запросы некоторого числа клиентских приложений (рис. 24). При обработке запроса сервер должен гарантировать актуальность информации, содержащейся в отсылаемом ответе на запрос, для чего предварительно нужно выполнить синхронизацию содержимого его собственной базы с другими серверами. Таким образом, серверу необходимо ждать полной синхронизации либо генерировать ответ на основе несинхронизированных данных. Возможен и третий вариант, когда по каким-либо причинам синхронизация производится только с частью серверов системы. В первом случае оказывается невыполненным требование по доступности, во втором — по согласованности, в третьем — по устойчивости к разделению [14].

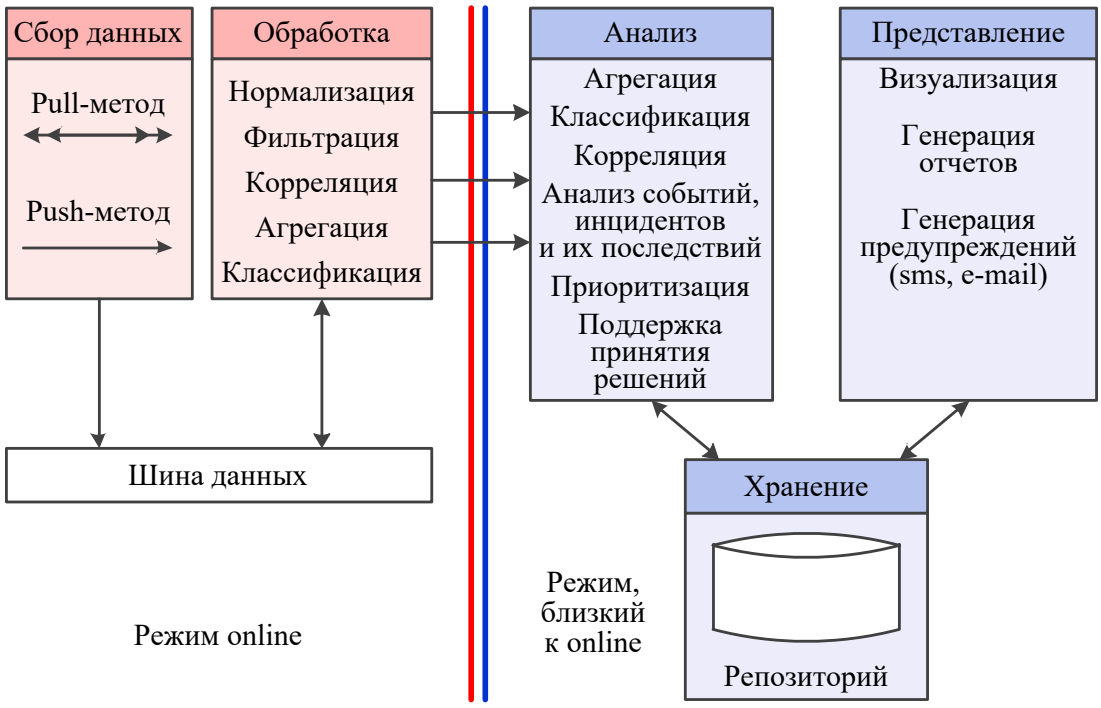


Рис. 24. Выполнение запроса к распределенной системе

Выводы по главе 2

Мы рассмотрели следующие модели данных:

– иерархическая модель;

– сетевая модель;

– реляционная модель;

– постреляционная модель;

– объектно-ориентированная модель;

– XML-данные.

К ограничениям иерархической модели данных можно отнести следующие:

– отсутствует явное разделение логических и физических характеристик модели;

– для представления неиерархических отношений данных требуются дополнительные манипуляции;

– непредвиденные запросы могут требовать реорганизации базы данных.

Достоинства иерархической модели:

– простота понимания и использования. Пользователи систем обработки данных хорошо знакомы с иерархическими структурами;

– обеспечение определенного уровня независимости данных;

– простота оценки операционных характеристик благодаря заранее заданным взаимосвязям.

Недостатки иерархической модели:

– отношение «многие-ко-многим» непосредственно не поддерживается; это основной недостаток иерархических моделей;

– из-за строгой иерархической упорядоченности объектов модели значительно усложняются операции включения и удаления;

– удаление исходных объектов влечет удаление порожденных. Поэтому выполнение операции УДАЛИТЬ требует особой осторожности;

– особенности иерархических структур обусловливают сложность операций манипулирования данными;

– корневой тип узла является главным. Доступ к любому порожденному узлу возможен только через исходный.

Достоинства сетевой модели данных:

– в рамках сетевых СУБД легко реализуются и иерархические даталогические модели;

– сетевые СУБД поддерживают сложные соотношения между типами данных, что делает их пригодными в различных приложениях. Однако пользователи таких СУБД ограничены связями, определенными для них разработчиками БД приложений;

– сетевые СУБД предполагают разработку БД приложений опытными программистами и системными аналитиками.

Недостаток сетевой модели — проблема обеспечения сохранности информации, решению которой уделяется повышенное внимание при проектировании сетевых БД.

Достоинства реляционной модели:

– простота представления данных;

– запросы не строятся на основе заранее определенной структуры — могут быть сформулированы на непроцедурном языке;

– независимость данных;

– реляционная модель хранения данных основана на хорошо проработанной теории отношений;

– при проектировании базы данных применяются строгие методы, построенные на использовании реляционной алгебры;

– простота внесения изменений в базу данных.

Недостатки реляционной модели:

– невозможность представления объектов с отношением «многие-ко-многим» в одной таблице;

– низкое быстродействие;

– требуется большой объем внешней памяти.

В ответ на недостатки иерархической, сетевой и реляционной моделей были разработаны постреляционные СУБД. Новые модели чаще всего начинают применяться к новым типам данных, которые старые технологии не были в состоянии адекватно анализировать. К примеру, классические методики, такие как предагрегирование, не могут гарантировать небольшое время ответа на запросы, если данные постоянно меняются, как это

происходит, например, когда информация поступает с датчиков или от движущихся объектов (автомобили, оснащенные средствами глобального позиционирования).

Также недостатками предшествующих моделей было обусловлено появление объектно-ориентированного подхода, обладающего следующими преимуществами в обеспечении целостности:

– автоматическое поддержание отношений наследования;

– возможность объявить некоторые поля данных и методы объекта «скрытыми», невидимыми для других объектов; такие поля и методы используются только для самого объекта;

– создание процедур контроля целостности внутри объекта.

Еще одной успешной реализацией постреляционного подхода можно считать концепцию XML-данных. XML-документы состоят из единиц размещения, называемых сущностями, которые содержат разобранные или неразобранные данные. Разобранные данные состоят из набора символов, одна часть которых образует символьные данные, а другая — разметку. Разметка образует описание схемы размещения и логической структуры документа. Язык XML дает механизм создания ограничений для указанной схемы размещения и логической структуры.

За последние два десятилетия рост объемов обрабатываемых данных, а также необходимость децентрализации их хранения и обработки подтолкнули к созданию новых моделей данных, получивших название «нереляционных» (NoSQL). Основные типы моделей NoSQL подробно рассмотрены в главе 3.

Вопросы для самоконтроля

1. Опишите основные компоненты и структуру иерархической модели данных.
2. Приведите основные понятия сетевой модели данных — элемент данных, агрегат данных, запись, ключ, набор данных. В чем сходство и отличие сетевой модели от иерархической?
3. Подробно опишите реляционную модель данных. Какие существуют подходы к проектированию реляционной базы данных?

4. Что такое «не первая нормальная форма» (NF2)? Назовите основные разновидности постреляционных систем.
5. Назовите основные принципы объектно-ориентированного подхода. Опишите структуру объектной модели. В чем трудности объектно-ориентированного моделирования данных?
6. Чем обусловлено появление XML-языка и XML-данных? Опишите структуру XML-документа. Каковы особенности языков XPath XQuery?
7. Назовите причины появления нереляционных СУБД (NoSQL).

Глава 3

Нереляционные СУБД

3.1. Основные модели данных

В распределенных системах управления данными используются два основных приема (практически всегда применяемых совместно) [15, с. 35]:

– разделение данных (шардинг, sharding) — подход, при котором каждый узел системы содержит свою часть данных и выполняет операции над ними. Этот метод является основным средством обеспечения горизонтальной масштабируемости, однако теперь операции над несколькими объектами могут вовлечь в работу несколько узлов, что требует активной передачи данных по сети. Кроме того, при увеличении числа узлов, хранящих данные, увеличивается вероятность сбоев, что требует наличия избыточности — применения репликации. Шардинг также порождает такие задачи, как распределение данных по узлам, балансировка нагрузки, оптимизация сетевых взаимодействий и т. д.;

– репликация (replication) — подход, при котором одни и те же данные хранятся на нескольких узлах в сети. Этот метод помогает повысить надежность системы и справляться как со сбоями отдельных узлов, так и с потерей целого кластера. Кроме того, репликация позволяет масштабировать операции чтения (несколько реже — записи). Серьезной задачей является поддержка согласованного состояния копий данных (реплик): при синхронном обновлении реплик увеличивается время ответа си-

стемы, а при асинхронном — возникает промежуток времени, когда реплики находятся в несогласованном состоянии.

Существуют две основные схемы репликации (поддерживающие как синхронные, так и асинхронные варианты) [15, с. 35–36]:

– ведущий-ведомый (master-slave) — при такой схеме репликации операции модификации данных обрабатывает только ведущий узел (master), а сделанные изменения синхронно или асинхронно передаются на ведомого (slave). Чтения могут осуществляться как с ведущего узла (гарантированно содержит последнюю версию данных), так и с ведомого (данные могут быть несколько устаревшими при асинхронной репликации и «отставать» от ведущего);

– ведущий-ведущий (master-master, multi-master) — при этой схеме все узлы могут обрабатывать операции записи и передавать обновления остальным. В этом случае реализовать синхронную репликацию достаточно сложно, к тому же резко возрастают задержки, связанные с сетевыми взаимодействиями. При асинхронном обновлении возникает другая проблема — могут появиться конфликтующие версии данных, которые требуют наличия механизма для определения и разрешения конфликтов (автоматически или на уровне приложения).

Обычно NoSQL-системы делят, исходя из модели данных, на следующие основные классы [15, с. 38]:

1) системы «ключ-значение» (Key-Value Stores);

2) документные СУБД (Document Stores);

3) системы типа Google BigTable (Extensible Record Stores / Wide Column Stores / Column Families).

Рассмотрим эти классы более подробно.

3.1.1. Системы «ключ-значение»

NoSQL-системы типа «ключ-значение» хранят данные (неструктурированные или структурированные) и позволяют иметь доступ к ним при помощи единственного уникального ключа. Работа с данными обычно осуществляется с помощью простых операций вставки, удаления и поиска по ключу. Вторичные ключи и индексы в таких системах не поддерживаются. При этом

может поддерживаться некоторая структура данных, позволяющая менять отдельные поля объекта, но не дающая строить по ним запросы (в этом заключается основное отличие систем типа «ключ-значение» от документных СУБД) [15, с. 38].

Это простой метод хранения данных, и он, как известно, хорошо масштабируется. Пара ключ-значение — это устоявшаяся концепция во многих языках программирования. Языки программирования обычно называют ключ-значение ассоциативным массивом или структурой данных. Ключ-значение также называют словарем или хешем [16].

Ниже в табл. 8 и 9 приведены примеры хранилищ ключевых значений.

Таблица 8

Пример хранилища значений (телефонный справочник)

Ключ	Значение
Сергей	904 5235678
Мария	902 5678901
Владимир	909 6789012
Никита	926 7890123

Таблица 9

Пример хранилища значений (таблица IP-адресации)

Ключ	Значение
202.45.12.34	01:23:36:0f:a2:33
202.45.123.4	00:25:33:da:4c:01
245.12.33.45	02:03:33:10:e2:b1
101.234.55.1	b8:67:a3:11:23:b1

Ключ в паре ключ-значение должен быть уникальным идентификатором. Он позволяет получить доступ к значению, связанному с этим ключом. Теоретически ключом может быть что угодно. Но это может зависеть от СУБД. Одна СУБД может накладывать ограничения, а другая — нет. Например, в Redis максимально допустимый размер ключа составляет 512 МБ. В качестве ключа можно использовать любую двоичную последовательность — от короткой строки текста до содержимого файла изображения. Даже пустая строка является допустимым

ключом. Однако по соображениям производительности следует избегать слишком длинного ключа. Но чересчур короткий текст также может вызвать проблемы с читабельностью. В любом случае ключ должен следовать согласованной конвенции [16].

Значение в хранилище ключ-значение может быть любым, таким как текст (длинный или короткий), число, код разметки, такой как HTML, программный код, такой как PHP, изображение и т. д. Значение также может быть списком или даже другой парой ключ-значение, инкапсулированной в объект. Некоторые СУБД хранилища ключей позволяют указать тип данных для значения. Например, можно указать, что значение должно быть целым числом. Другие СУБД не предоставляют этой функции, и поэтому значение может быть любого типа. Например, СУБД Redis позволяет указать следующие типы данных [16]:

– бинарно-безопасные строки;

– списки — коллекции строковых элементов, отсортированных в соответствии с порядком вставки;

– наборы — коллекции уникальных несортированных строковых элементов;

– сортированные наборы, похожие на наборы, но где каждый строковый элемент связан с плавающим числовым значением, называемым оценкой. Позволяет, например, выбрать топ-10 или нижние 10 и т. д.;

– хеши — карты, состоящие из полей, связанных со значениями. И поле, и значение являются строками;

– битовые массивы (или просто растровые изображения);

– структуры HyperLogLogs — вероятностные структуры данных, которые используются для оценки мощности множества.

Базы данных ключ-значение могут быть применены ко многим сценариям. Например, хранилища ключ-значение могут быть использованы для хранения [16]:

1) в области веб-сайтов — профили пользователей; информация о сеансе; комментарии к статье/блогу; электронные письма; сообщения о состоянии;

2) в области электронной коммерции — содержимое корзины покупок; категории продуктов; детали продукта; отзывы о продукте;

3) в сфере сетевого обслуживания данных — телекоммуникационные справочники; таблицы IP-адресации; дедупликация данных.

Базы данных ключ-значение могут даже хранить целые веб-страницы, используя URL-адрес в качестве ключа и веб-страницу в качестве значения. Однако в зависимости от того, что нужно сделать с данными, другие модели БД могут быть более подходящими для хранения веб-страниц и статей.

3.1.2. Документные СУБД

Документная СУБД (также известная как ориентированная на документ база данных, агрегированная база данных или просто хранилище документов или база данных документов) — это база данных, которая использует ориентированную на документ модель для хранения данных. Документные СУБД хранят каждую запись и связанные с ней данные в одном документе. Каждый документ содержит полуструктурированные данные, которые могут быть запрошены с помощью различных инструментов запросов и аналитики СУБД [17].

Вот два примера документов, которые могут храниться в документной СУБД. В обоих примерах используются одни и те же данные — просто они написаны на разных языках [17]:

Первый пример написан в формате XML.

```
<artist>
  <artistname>Iron Maiden</<artistname>
  <albums>
    <album>
      <albumname>The Book of Souls</albumname>
      <datereleased>2015</datereleased>
      <genre>Hard Rock</genre>
    </album>
    <album>
      <albumname>Killers</albumname>
      <datereleased>1981</datereleased>
      <genre>Hard Rock</genre>
    </album>
```

```
    <album>
      <albumname>Powerslave</albumname>
      <datereleased>1984</datereleased>
      <genre>Hard Rock</genre>
    </album>
    <album>
      <albumname>Somewhere in Time</albumname>
      <datereleased>1986</datereleased>
      <genre>Hard Rock</genre>
    </album>
  </albums>
</artist>
```

Второй пример написан на языке JSON.

```
{
   '_id' : 1,
   'artistName' : { 'Iron Maiden' },
   'albums' : [
      {          'albumname' : 'The Book of Souls',
         'datereleased' : 2015,
         'genre' : 'Hard Rock'},
{        'albumname' : 'Killers',
         'datereleased' : 1981,
         'genre' : 'Hard Rock'},
{        'albumname' : 'Powerslave',
         'datereleased' : 1984,
         'genre' : 'Hard Rock'},
{        'albumname' : 'Somewhere in Time',
         'datereleased' : 1986,
         'genre' : 'Hard Rock'}
   ]
}
```

Если бы мы ввели вышеуказанные данные в реляционную базу данных, информация обычно хранилась бы в трех разных таблицах — с отношением, связывающим их вместе через поля первичного ключа и внешнего ключа (табл. 10, 11, 12).

Таблица 10

Таблица «Исполнитель»

Идентификатор исполнителя	Имя исполнителя
1	Iron Maiden
2	Devin Townsend
3	The Wiggles
4	…

Таблица 11

Таблица «Альбом»

Идентификатор альбома	Название альбома	Дата выпуска	Идентификатор исполнителя	Идентификатор жанра
1	The Book of Souls	2015	1	3
2	Killers	1981	1	3
3	Powerslave	1984	1	3
4	Somewhere in Time	1986	1	3
5	Ziltoid the Omniscient	2007	2	3
6	…	…	…	…

Таблица 12

Таблица «Жанр»

Идентификатор жанра	Название жанра
1	Country
2	Blues
3	Hard Rock
4	…

Вышеуказанные три таблицы были бы связаны ключами Идентификатор исполнителя и Идентификатор жанра.

Таким образом, можно обозначить некоторые существенные различия между базами данных хранилища документов и реляционными базами данных [17].

Реляционные базы данных хранят данные в нескольких таблицах, каждая таблица содержит столбцы, а каждая строка представляет запись. Информация о любой данной сущности может быть распределена по множеству таблиц. Данные из разных таблиц могут быть связаны только путем установления связи между таблицами.

В свою очередь, базы данных документов не используют таблицы как таковые. Они хранят все данные об объекте в одном документе.

В случае реляционных баз данных перед загрузкой данных необходимо создать схему. С базами данных хранилища документов (и большинством других баз данных NoSQL) нет такого требования. Вы можете просто пойти дальше и загрузить данные без какой-либо предопределенной схемы.

Таким образом, в хранилище документов любые два документа могут содержать различную структуру и тип данных. Например, если один пользователь решит не указывать свою дату рождения, это даже не будет полем в документе. Если другой пользователь укажет свою дату рождения, это будет поле в этом документе. Если бы это была реляционная база данных, дата рождения все равно была бы полем для обоих пользователей — она просто не содержала бы значения [17].

Базы данных документов могут очень хорошо масштабироваться по горизонтали. Данные могут храниться на многих тысячах компьютеров, и система будет работать хорошо. Это часто называют шардингом. Реляционные базы данных не очень хорошо подходят для масштабирования таким способом. Реляционные БД больше подходят для вертикального масштабирования (добавления дополнительной памяти, хранилища и т. д.). Поскольку существует предел тому, сколько ресурсов вы можете вместить в одну машину, может наступить момент, когда горизонтальное масштабирование станет единственным вариантом.

У хранилищ документов нет внешних ключей, как у реляционных баз данных. Внешние ключи используются реляционными базами данных для обеспечения связи между таблицами. Если необходимо установить связь с базой данных документов, это должно быть сделано на уровне приложения. Однако вся идея модели документа заключается в том, что любые данные, связанные с записью, хранятся в одном документе. Поэтому необходимость установления связи при использовании модели документа не должна быть столь распространенной, как в реляционной базе данных [17].

Большинство реляционных баз данных используют SQL в качестве стандартного языка запросов. Базы данных хранили-

ща документов, как правило, используют другие языки запросов (хотя некоторые из них созданы для поддержки SQL). Многие базы данных документов могут быть запрошены с помощью таких языков, как XQuery, XSLT, SPARQL, Java, JavaScript, Python и т. д.

Базы данных документов похожи на базы данных ключ-значение тем, что есть ключ и значение. Данные хранятся в виде значения. Связанный с ним ключ является уникальным идентификатором этого значения.

Разница заключается в том, что в базе данных документов значение содержит структурированные или полуструктурированные данные. Это структурированное/полуструктурированное значение называется документом. Структурированные/полуструктурированные данные, составляющие документ, могут быть закодированы с использованием любого количества методов, включая XML, JSON, YAML, BSON и т. д. Он также может быть закодирован с помощью двоичных файлов, таких как PDF-файлы, документы MS Office и т. д. [17].

Одно из преимуществ баз данных хранилища документов по сравнению с базами данных ключ-значение заключается в том, что вы можете запрашивать сами данные. Вы можете запросить структуру документа, а также элементы внутри этой структуры. Поэтому вы можете вернуть только те части документа, которые вам необходимы.

С базой данных ключ-значение вы получаете все значение — независимо от того, насколько большим (и, казалось бы, структурированным) оно может быть. Вы не можете сделать запрос в пределах значения.

Документоориентированные базы данных хорошо подходят для широкого спектра вариантов использования, например:

1) в области веб-приложений — системы управления контентом; блог-платформы; приложения для электронной коммерции; веб-аналитика; данные о предпочтениях пользователя;

2) в области пользовательского контента — чат-занятия; твиты; сообщения в блоге; рейтинги; комментарии;

3) в области каталогов данных — учетные записи пользователей; каталоги продукции; реестры устройств для Интернета вещей; BoM-системы;

4) в игровой индустрии — внутриигровая статистика; интеграция в социальные сети; таблицы лидеров с высокими баллами; сообщения внутриигрового чата; членство в гильдии игроков; выполненные задачи;

5) в сфере сетевого обслуживания данных — данные датчиков с мобильных устройств; файлы журналов; аналитика в реальном времени; иные данные Интернета вещей.

3.1.3. «Большая таблица»

Большая таблица — это тип базы данных, которая хранит данные с использованием модели, ориентированной на столбцы. Она также может называться [18]: база данных столбцов; база данных семейства столбцов; база данных, ориентированная на столбцы; широкое хранилище столбцов; столбчатая база данных; столбчатое хранилище.

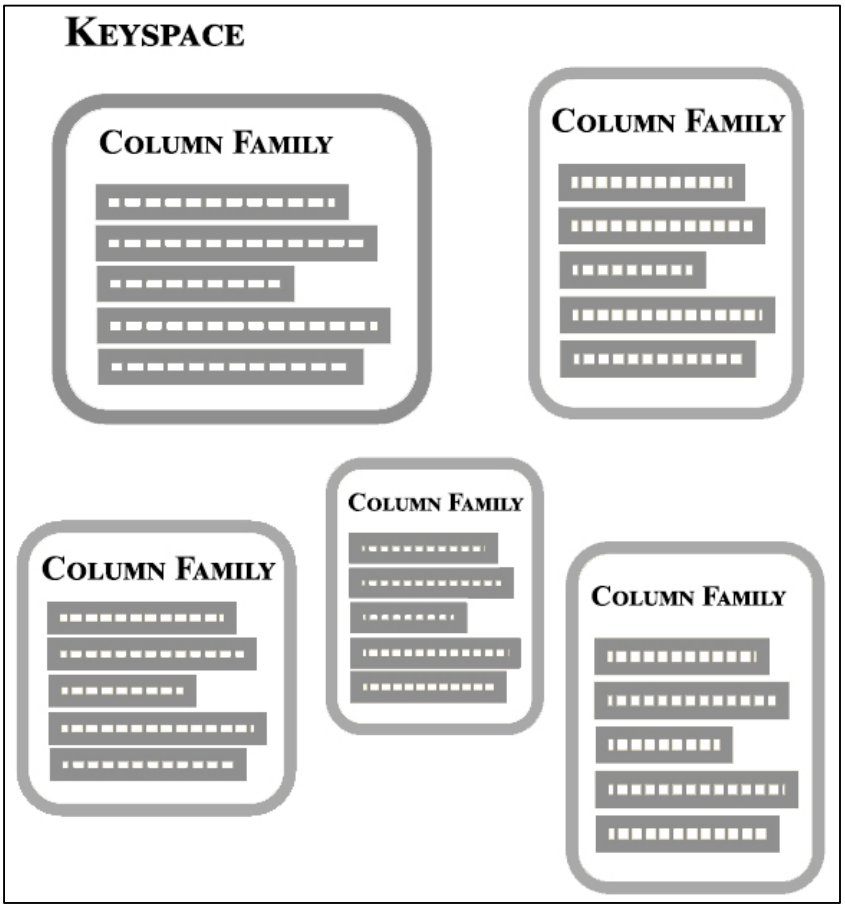


Рис. 25. Пространство ключей, содержащее семейства столбцов

Базы данных столбцов используют концепцию, называемую пространством ключей. Пространство ключей — это что-то вроде схемы в реляционной модели. Пространство ключей содержит все семейства столбцов (наподобие таблиц в реляционной модели), которые содержат строки, в свою очередь содержащие столбцы (рис. 25).

Пример семейства столбцов показан на рис. 26.

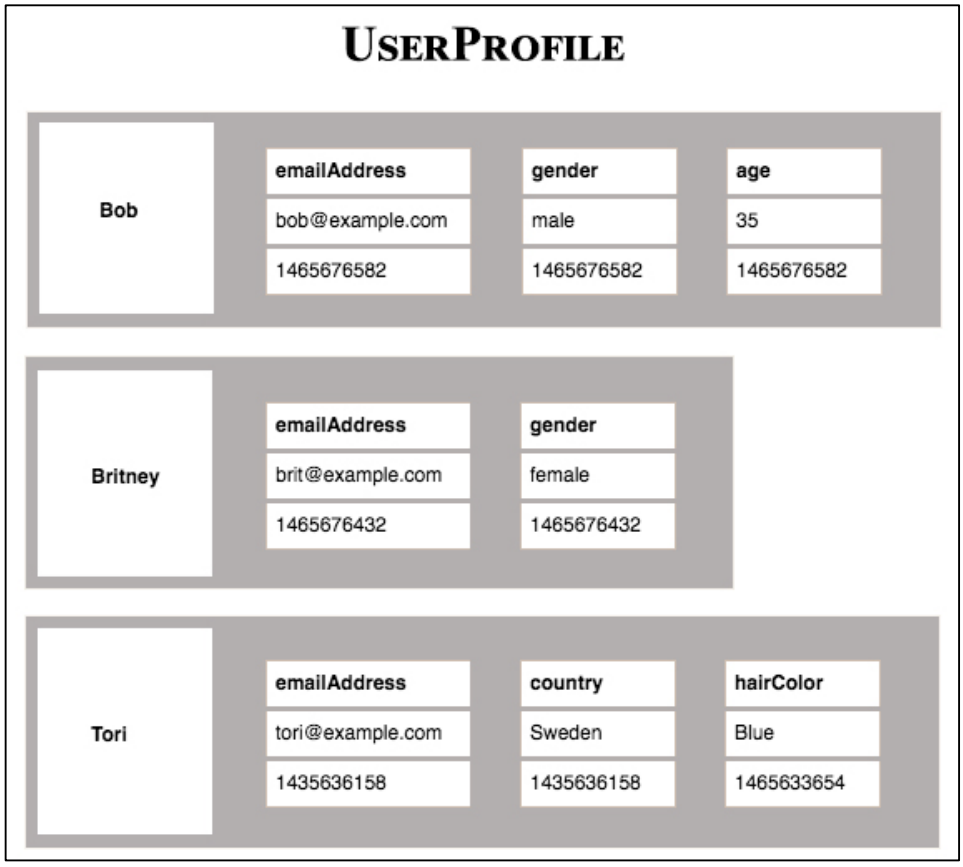


Рис. 26. Пример семейства столбцов, содержащего три строки, каждая из которых содержит набор столбцов

Как показано на рис. 26 [18]:

– семейство столбцов состоит из нескольких строк;

– каждая строка может содержать разное количество столбцов по сравнению с другими строками. И столбцы не должны совпадать со столбцами в других строках (они могут иметь разные имена, типы данных и т. д.);

– каждый столбец содержится в своей строке. Он не охватывает все строки, как в реляционной базе данных. Каждый столбец содержит пару имя/значение, а также метку времени.

Структура строки показана на рис. 27.

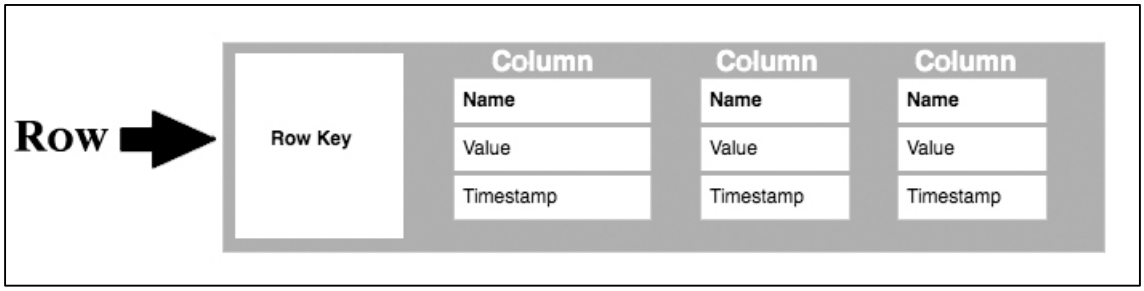


Рис. 27. Структура строки

Согласно рис. 27 в строке содержатся элементы:

– ключ строки. Каждая строка имеет ключ, который является ее уникальным идентификатором;

– колонка. Каждый столбец содержит имя, значение и метку времени;

– имя. Это имя пары имя/значение;

– ценность. Это значение пары имя/значение;

– отметка времени. Указывает дату и время вставки данных. Это можно использовать для определения самой последней версии данных.

Некоторые СУБД расширяют концепцию семейства столбцов, чтобы обеспечить дополнительную функциональность/возможность хранения. Например, у Cassandra есть концепция составных столбцов, которые позволяют вам вкладывать объекты внутри столбца. Вот некоторые ключевые преимущества столбчатых баз данных [18]:

– компрессия. Хранилища столбцов очень эффективны при сжатии и/или секционировании данных;

– запросы агрегации. Благодаря своей структуре столбчатые базы данных особенно хорошо работают с запросами агрегации (такими как SUM, COUNT, AVG и т. д.);

– масштабируемость. Столбчатые базы данных очень масштабируемы. Они хорошо подходят для массовой параллельной обработки (MPP), которая включает в себя распространение данных по большому кластеру машин, часто тысячи;

– быстрая загрузка и запрос. Столбчатые базы данных могут быть загружены чрезвычайно быстро. Таблица в миллиард

строк может быть загружена в течение нескольких секунд. Вы можете начать анализировать практически сразу.

Это лишь некоторые из преимуществ, которые делают столбчатые базы данных популярным выбором для организаций, работающих с большими данными.

3.2. Наиболее популярные нереляционные СУБД

Redis

Redis — это хранилище структур данных с открытым исходным кодом (лицензия BSD), используемое в качестве базы данных, кеша и брокера сообщений. Redis поддерживает такие структуры данных, как строки, хеши, списки, наборы, сортированные наборы с запросами диапазона, растровые изображения, гиперлоги, геопространственные индексы и потоки. Redis имеет встроенную репликацию, сценарии Lua, выселение LRU, транзакции и различные уровни персистентности на диске, а также обеспечивает высокую доступность через Redis Sentinel и автоматическое секционирование с помощью Redis Cluster [19].

Можно выполнять атомарные операции над этими типами, такие как добавление к строке; увеличение значения в хеше; перемещение элемента в список; вычисление пересечения множества, объединения и разности; получение члена с самым высоким рангом в отсортированном наборе. Для достижения максимальной производительности Redis работает с набором данных в памяти. В зависимости от варианта использования можно сохранять свои данные, либо периодически сбрасывая набор данных на диск, либо добавляя каждую команду в дисковый журнал. Redis также поддерживает асинхронную репликацию с очень быстрой неблокирующей первой синхронизацией, автоматическим переподключением с частичной ресинхронизацией на сетевом разделении [19].

Redis написан на ANSI C и работает в большинстве POSIX-систем, таких как Linux, *BSD и OS X, без внешних зависимостей. Linux и OS X — это две операционные системы, в которых Redis разрабатывается и тестируется больше всего, рекомендуется использовать Linux для развертывания. Redis мо-

жет работать в системах, производных от Solaris, таких как SmartOS, но «поддержка — это лучшее усилие». Официальной поддержки сборок Windows нет.

Voldemort

Voldemort — это распределенная система хранения ключей и ценностей, обладающая следующими свойствами [20]:

– данные автоматически реплицируются на несколько серверов;

– данные автоматически секционируются, поэтому каждый сервер содержит только подмножество общих данных;

– обеспечивается настраиваемая согласованность (строгий кворум или возможная согласованность);

– сбой сервера мгновенно прослеживается;

– подключаемые механизмы хранения данных — BDB-JE, MySQL, Read-Only;

– подключаемая сериализация — Protocol Buffers, Thrift, Avro and Java Serialization;

– элементы данных версируются для обеспечения максимальной целостности данных в сценариях сбоев без ущерба для доступности системы;

– каждый узел независим от других узлов, отсутствует центральная точка отказа или координации;

– хорошая производительность одного узла: до 10–20 тыс. операций в секунду в зависимости от машин, сети, дисковой системы и коэффициента репликации данных;

– поддерживаются подключаемые стратегии размещения данных для распределения между центрами обработки данных, географически удаленными друг от друга.

Преимущества Voldemort [20]:

– объединяет кеширование в памяти с системой хранения, так что отдельный уровень кеширования не требуется;

– в отличие от репликации MySQL, как чтение, так и запись масштабируются горизонтально;

– порционирование данных является прозрачным и предоставляет возможности для расширения кластера без перебалансировки всех данных;

– репликация и размещение данных решаются простым API, чтобы иметь возможность адаптировать широкий спектр специфичных для приложения стратегий;

– уровень хранения гибкий, поэтому разработка и модульное тестирование могут быть выполнены в черновике памяти системы хранения без необходимости использования реального кластера (или даже реальной системы хранения);

– исходный код доступен под лицензией Apache 2.0.

Aerospike

Aerospike — это распределенная масштабируемая база данных. Архитектура имеет три ключевые цели [21]:

– создание гибкой, масштабируемой платформы для веб-приложений;

– обеспечение надежности и безотказности (как в ACID) на уровне, не уступающем традиционным базам данных;

– обеспечение оперативной эффективности при минимальном ручном участии.

Как было сказано в Трудах VLDB (Very Large Databases), архитектура Aerospike состоит из трех слоев [21]:

1) клиентский уровень. Этот кластерно-ориентированный уровень включает клиентские библиотеки с открытым исходным кодом, которые реализуют API Aerospike, отслеживают узлы и знают, где находятся данные в кластере;

2) уровень кластеризации и распределения данных. Этот уровень управляет связью между кластерами и автоматизирует отказоустойчивость, репликацию, репликацию между центрами обработки данных (XDR), а также интеллектуальную балансировку и миграцию данных;

3) уровень хранения данных. Этот уровень обеспечивает надежное хранение данных в DRAM и флеш-памяти для быстрой загрузки.

Клиентский уровень [21]:

– реализует API Aerospike, протокол клиент-сервер и взаимодействует непосредственно с кластером;

– отслеживает узлы и знает, где хранятся данные, мгновенно узнавая об изменениях конфигурации кластера, о доступности и недоступности узлов;

– реализует свой собственный пул соединений TCP/IP для повышения эффективности. Также обнаруживает сбои транзакций, которые не достигли уровня сбоев узлов в кластере, и перенаправляет эти транзакции на узлы с копиями данных;

– отправляет запросы непосредственно на узел с данными и повторно перенаправляет запросы по мере необходимости (например, во время переконфигурации кластера).

Архитектура клиентского уровня уменьшает задержку транзакций, разгружает работу из кластера и устраняет сложности для разработчика. Она гарантирует, что приложения не должны будут перезапускаться в случае недоступности узлов.

Уровень распределения данных предназначен для устранения ручных операций с систематической автоматизацией всех функций управления кластером. Он включает в себя три модуля [21]:

1) модуль управления кластером. Отслеживает узлы в кластере;

2) модуль миграции данных. При добавлении или удалении узлов устанавливается принадлежность к кластеру базы данных Aerospike. Каждый узел использует распределенный хеш-алгоритм для разделения первичного индексного пространства на срезы данных и назначения владельцев. Модуль миграции данных Aerospike разумно балансирует распределение данных по всем узлам кластера, гарантируя, что каждый бит данных реплицируется во всех узлах кластера и центрах обработки данных. Эта операция задается в конфигурации фактора репликации системы;

3) модуль обработки транзакций. Считывает и записывает данные по запросу, а также обеспечивает гарантии согласованности и изоляции. Этот модуль отвечает за:

– синхронную/асинхронную репликацию — для записи с немедленной согласованностью она распространяет изменения на все реплики перед фиксацией данных и возвращением результата клиенту;

– прокси-сервер — в редких случаях при перенастройке кластера, когда клиентский уровень может ненадолго устареть, модуль обработки транзакций прозрачно передает запрос другому узлу;

– разрешение дубликатов данных — для кластеров, восстановленных после секционирования (в том числе при перезапуске узлов), этот модуль разрешает любые конфликты между различными копиями данных. Разрешение может быть основано либо на количестве поколений (версии), либо на времени последнего обновления.

Уровень хранения данных. Потоки данных поступают в контейнеры, пространства имен, которые семантически подобны базам данных в реляционных СУБД. В пространстве имен данные подразделяются на наборы (таблицы СУБД) и записи (строки СУБД). Каждая запись имеет индексированный ключ, уникальный в наборе, и одну или несколько именованных ячеек (столбцов СУБД), которые содержат значения, связанные с записью. При этом нет необходимости определять наборы и ячейки — для максимальной гибкости они могут быть добавлены во время выполнения. Значения в ячейках строго типизированы и могут включать любой поддерживаемый тип данных. Ячейки не типизированы, поэтому разные записи могут иметь одну и ту же ячейку со значениями разных типов [21].

Индексы, включая первичный индекс и необязательные вторичные индексы, по умолчанию хранятся в DRAM для сверхбыстрого доступа. Первичный индекс также может быть сконфигурирован для хранения в постоянной памяти или на флеш-устройстве. Значения могут быть сохранены в DRAM или с меньшими затратами на твердых накопителях.

Таким образом, в Aerospike [21]:

– 100 миллионов ключей занимают всего 6,4 ГБ. Хотя ключи не имеют ограничений по размеру, каждый ключ эффективно хранится всего в 64 байтах;

– чтобы минимизировать задержку, запись на диск выполняется большими блоками. Этот механизм обходит стандартную файловую систему, традиционно настроенную на вращающиеся жесткие диски;

– интеллектуальный дефрагментатор и выселитель работают вместе, чтобы гарантировать, что в DRAM есть место, а данные никогда не теряются и всегда безопасно записываются на диск. Интеллектуальный дефрагментатор отслеживает количество активных записей в каждом блоке и восстанавливает бло-

ки, которые находятся ниже минимального уровня использования. Выселитель удаляет просроченные записи и восстанавливает память, если система выходит за установленную (высокую) отметку. Время истечения срока действия настраивается для каждого пространства имен. Возраст записи рассчитывается с момента последней модификации. Приложение может переопределить время жизни по умолчанию и указать, что запись никогда не должна быть удалена.

MongoDB

Запись в MongoDB — это документ, представляющий собой структуру данных, состоящую из пар полей и значений. Документы MongoDB похожи на объекты JSON. Значения полей могут включать в себя другие документы, массивы и массивы документов (рис. 28) [22].

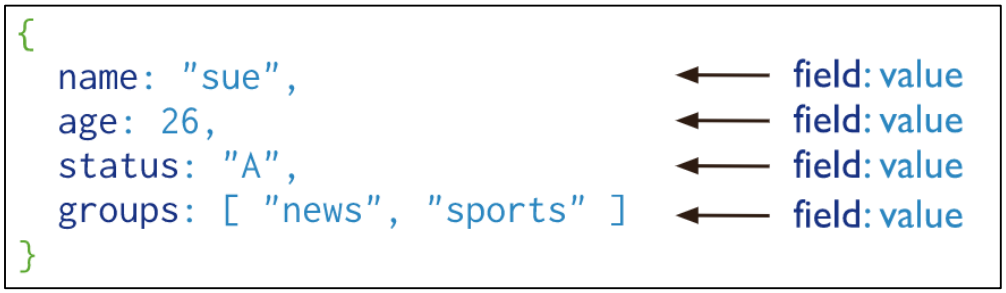


Рис. 28. MongoDB — значения полей

Преимущества использования документов заключаются в следующем [22]:

– документы (объекты) соответствуют собственным типам данных во многих языках программирования;

– встроенные документы и массивы уменьшают потребность в дорогостоящих соединениях;

– динамическая схема поддерживает свободный полиморфизм;

– коллекции / представления / материализованные представления — по требованию;

– MongoDB хранит документы в коллекциях. Коллекции аналогичны таблицам в реляционных базах данных.

Основные характеристики [22]:

1) высокая производительность. MongoDB обеспечивает высокую производительность сохранения данных, в частности:

– поддержка встроенных моделей данных снижает активность ввода-вывода в системе баз данных;

– индексы поддерживают более быстрые запросы и могут включать ключи из встроенных документов и массивов;

2) расширенный язык запросов, используемый для поддержки операций чтения и записи (CRUD), включая:

– агрегирование данных;

– текстовый поиск и геопространственные запросы;

3) высокая степень доступности. Средство репликации MongoDB, называемое набором реплик (группа серверов MongoDB, которые поддерживают один и тот же набор данных, обеспечивая избыточность и повышая доступность данных), обеспечивает автоматическое резервирование данных при отказе;

4) горизонтальная масштабируемость. MongoDB обеспечивает горизонтальную масштабируемость как часть своей основной функциональности:

– сегментация распределяет данные по кластеру машин;

– создание зон данных поддерживается на основе остаточного ключа. В сбалансированном кластере MongoDB направляет чтение и запись, охватываемые зоной, только на части внутри зоны;

5) поддержка нескольких механизмов хранения данных:

– механизм хранения WiredTiger (включая поддержку шифрования в состоянии покоя);

– механизм хранения данных в памяти;

– MongoDB предоставляет подключаемый API-интерфейс, который позволяет третьим сторонам разрабатывать механизмы хранения данных для MongoDB.

OrientDB

OrientDB — это первая многомодельная СУБД с открытым исходным кодом, которая сочетает в себе мощь графиков и гибкость документов в одной масштабируемой, высокопроизводительной операционной базе данных. Многомодельные базы данных признают необходимость использования нескольких мо-

делей данных, объединяя их для снижения операционной сложности и поддержания согласованности данных [23].

Ядро OrientDB поддерживает графические, документные, объектные модели, а также модель «ключ-значение», поэтому OrientDB можно использовать в качестве замены продукта в любой из этих категорий. Однако основное преимущество заключается в многомодельных возможностях СУБД, которые объединяют все функции четырех моделей в ядро. Это также является основным отличием от других многомодельных СУБД, поскольку они реализуют дополнительный уровень с API, который имитирует дополнительные модели, однако ядро составляет только одна модель [24]. Разберем модели Orient DB подробнее.

1. Документная модель. Данные в этой модели хранятся внутри документов. Документ — это набор пар ключ-значение (также называемых полями или свойствами), где ключ позволяет получить доступ к его значению. Значения могут содержать примитивные типы данных, встроенные документы или массивы других значений. OrientDB использует понятия «классы» и «кластеры» в качестве своей формы «коллекций» для группировки документов. Модель документов OrientDB также добавляет понятие «связь» как отношение между документами. С помощью OrientDB вы можете решить, вставлять ли документы или ссылаться на них напрямую. Когда вы извлекаете документ, все ссылки автоматически разрешаются. Это существенное отличие от других баз данных документов, в которых разработчик должен самостоятельно обрабатывать любые отношения между документами. В табл. 13 проведено сравнение между реляционной моделью, документной моделью и документной моделью OrientDB [24].

Таблица 13

Сравнительная характеристика реляционной модели, документной модели и документной модели OrientDB

Реляционная модель данных	Документная модель	Документная модель OrientDB
Таблица	Коллекция	Класс или кластер
Строка	Документ	Документ
Столбец	Пара ключ-значение	Поле документа
Отношение	–	Связь

2. Графовая модель. Граф представляет собой сетеподобную структуру, состоящую из вершин (узлов), соединенных между собой ребрами (дугами). Графовая модель OrientDB представлена концепцией графа свойств, которая определяет следующие понятия.

Вершина — сущность, которая может быть связана с другими вершинами и обладает следующими обязательными свойствами:

а) уникальный идентификатор;

б) набор входящих ребер;

в) набор исходящих ребер.

Ребро — сущность, связывающая две вершины и обладающая следующими обязательными свойствами:

а) уникальный идентификатор;

б) ссылка на входящую вершину (голову);

в) ссылка на исходящую вершину (хвост);

г) метка, определяющая тип связи/отношения между головной и хвостовой вершинами.

В дополнение к обязательным свойствам каждая вершина или ребро также может содержать набор пользовательских свойств. В табл. 14 представлено сравнение между графовой моделью, реляционной моделью данных и графовой моделью OrientDB [24].

Таблица 14

Сравнительная характеристика графовой модели, реляционной модели данных и графовой модели OrientDB

Реляционная модель данных	Графовая модель	Графовая модель OrientDB
Таблица	Классы вершина и ребро	Класс «V» (для вершины) и «E» (для ребра)
Строка	Вершина	Вершина
Столбец	Свойство вершины и ребра	Свойство вершины и ребра
Отношение	Ребро	Ребро

3. Модель «ключ-значение». Доступ к данным осуществляется через ключ, а значения могут быть простыми и сложными типами. OrientDB поддерживает документы и элементы гра-

фика в качестве значений, что позволяет создать более сложные структуры, чем классическая модель «ключ-значение». В табл. 15 проводится сравнение между реляционной моделью, моделью «ключ-значение» и моделью «ключ-значение» OrientDB [24].

Таблица 15

Сравнительная характеристика реляционной модели, модели «ключ-значение» и модели «ключ-значение» OrientDB

Реляционная модель данных	Модель «ключ-значение»	Модель «ключ-значение» OrientDB
Таблица	Контейнер	Класс или кластер
Строка	Пара ключ-значение	Документ
Столбец	–	Поле документа или свойство вершины и ребра
Отношение	–	Связь

4. Объектная модель [24]. Эта модель поддерживает наследование между типами, полиморфизм при обращении к базовому классу и прямое обращение от/к объектам. Сравнение между реляционной моделью, объектной моделью и объектной моделью OrientDB показано в табл. 16.

Таблица 16

Сравнительная характеристика реляционной модели, объектной модели и объектной модели OrientDB

Реляционная модель данных	Объектная модель	Объектная модель OrientDB
Таблица	Класс	Класс или кластер
Строка	Объект	Документ
Столбец	Свойство объекта	Поле документа или свойство вершины и ребра
Отношение	Указатель	Связь

Apache Cassandra

Apache Cassandra — это распределенная нереляционная база данных с открытым исходным кодом. Представляет собой секционированную модель хранения широких столбцов с последовательной семантикой. Изначально была разработана в Facebook с использованием поэтапной событийной архитектуры

(SEDA) для реализации комбинации методов распределенного хранения и репликации Amazon Dynamo в сочетании с моделью «большая таблица» от Google. Dynamo и Bigtable были разработаны для удовлетворения новых требований к масштабируемым, надежным и высокодоступным системам хранения данных, но у каждого из них были области, которые можно было улучшить [25].

Cassandra была разработана как комбинация обеих систем для удовлетворения возникающих крупномасштабных требований к хранению данных и объему запросов. Поскольку приложения стали требовать полной глобальной репликации и всегда доступных операций чтения и записи с низкой задержкой, стало необходимым разработать новый тип модели базы данных. Cassandra преследует следующие цели проектирования [25]:

– полная репликация баз данных, управляемых из нескольких источников;

– глобальная доступность с низкой задержкой;

– масштабирование на аутсерсном оборудовании;

– улучшение линейной пропускной способности с каждым дополнительным процессором;

– онлайн-балансировка нагрузки и рост кластеров;

– секционированные запросы, ориентированные на ключ;

– гибкая схема.

Свойства Cassandra [25]:

1) Cassandra предоставляет Cassandra Query Language (CQL), SQL-подобный язык, для создания и обновления схемы базы данных и доступа к данным. CQL позволяет пользователям организовывать данные в кластере узлов Cassandra с помощью следующих инструментов:

– пространство ключей — определяет, как реплицируется набор данных, например, в каких центрах обработки данных и сколько копий. Пространство ключей содержат таблицы;

– таблица — определяет типизированную схему для коллекции разделов. Таблицы Cassandra имеют гибкое добавление новых столбцов в таблицы с нулевым временем простоя. Таблицы содержат сегменты, которые содержат разделы, которые содержат столбцы;

– раздел — определяет обязательную часть первичного ключа, который должны иметь все строки в Cassandra. Все исполняемые запросы содержат в себе ключ раздела;

– строка — содержит набор столбцов, идентифицируемых уникальным первичным ключом, состоящим из ключа раздела и, возможно, дополнительных ключей кластеризации;

– столбец — единица данных с типом, принадлежащим строке;

2) CQL поддерживает множество расширенных функций для работы над секционированным набором данных, таких как:

– облегченные транзакции с одним разделом и атомарной семантикой сравнения и набора;

– пользовательские типы, функции и агрегаты;

– типы коллекций, включая наборы, карты и списки;

– локальные вторичные индексы;

– (экспериментальные) материализованные представления.

Cassandra предпочитает прямо не применять операции, требующие координации между разделами, поскольку они обычно медленны и плохо обеспечивают высокодоступную глобальную семантику. Например, Cassandra не поддерживает:

– перекрестные транзакции;

– распределенные соединения;

– внешние ключи или ссылочную целостность.

Параметры конфигурации Apache Cassandra настраиваются в файле cassandra.yaml, который можно редактировать вручную или с помощью инструментов управления конфигурацией. Некоторыми настройками можно управлять в режиме реального времени с помощью онлайн-интерфейса, другие требуют перезапуска базы данных, чтобы они вступили в силу. Кроме того, Cassandra поддерживает готовую функцию атомарного моментального снимка, которая представляет собой моментальный снимок данных Cassandra в определенный момент времени для легкой интеграции со многими инструментами резервного копирования. Cassandra также поддерживает инкрементные резервные копии, где данные могут быть сохранены по мере их записи [25].

Apache Druid

Apache Druid — это аналитическая база данных реального времени, предназначенная для быстрой аналитики срезов и кубов («OLAP» запросы) на больших наборах данных. Druid чаще всего используется в случаях, когда необходимы обработка в реальном времени, быстрая производительность запросов и высокая степень надежности. Таким образом, Druid обычно используется для поддержки графических интерфейсов аналитических приложений или в качестве бэкэнда для высококонкурентных API, требующих быстрой агрегации. Druid лучше всего работает с событийно-ориентированными данными [26].

Области применения Druid [26]:

– веб- и мобильная аналитика;

– сетевая телеметрическая аналитика (мониторинг производительности сети);

– хранение метрик сервера;

– аналитика цепочки поставок (производственные показатели);

– показатели производительности приложений;

– цифровой маркетинг/рекламная аналитика;

– бизнес-аналитика / OLAP.

Druid имеет многопроцессорную распределенную архитектуру, которая предназначена для работы в облаке и проста в эксплуатации. Каждый тип процесса Druid может быть настроен и масштабирован независимо, что дает максимальную гибкость в работе с кластером. Эта конструкция также обеспечивает повышенную отказоустойчивость: выход из строя одного компонента не будет прямо влиять на другие компоненты [27].

Druid имеет несколько типов процессов [27]:

– процессы-координаторы управляют доступностью данных в кластере;

– процессы — контроллеры перегрузок контролируют распределение нагрузки потребления данных;

– процессы-брокеры обрабатывают запросы от внешних клиентов;

– процессы-маршрутизаторы — необязательные процессы, которые могут направлять запросы процессам-брокерам, процессам-координаторам и процессам — контроллерам перегрузок;

– исторические процессы хранят запрашиваемые данные;

– процессы MiddleManager отвечают за прием данных.

Процессы Druid могут быть развернуты любым удобным способом, но рекомендуется организовать их в трех типах серверов:

– мастер — запускает процессы-координаторы и процессы — контроллеры перегрузок, управляет доступностью и приемом данных;

– запрос — запускает процессы-брокеры и процессы-маршрутизаторы, обрабатывает запросы от внешних клиентов;

– данные — запускает исторические процессы и процессы MiddleManager, выполняет рабочие нагрузки приема и хранит все запрашиваемые данные.

В дополнение к встроенным типам процессов Druid также имеет три типа внешних зависимостей (это дает возможность использовать существующую инфраструктуру) [27].

1. Глубокое хранение. Общее хранилище файлов, доступное каждому серверу Druid. В кластерном развертывании это обычно распределенное хранилище объектов, такое как S3 или HDFS, или сетевая файловая система. В развертывании с одним сервером, как правило, это будет локальный диск. Druid использует глубокое хранилище для хранения любых данных, которые были поглощены системой.

Druid использует глубокое хранение только как резервную копию данных и как способ передачи данных в фоновом режиме между процессами Druid. Чтобы ответить на запросы, исторические процессы обращаются к глубокому хранилищу, а читают предварительно выбранные сегменты со своих локальных дисков до того, как будут обработаны какие-либо запросы. Это означает, что Druid не нуждается в доступе к глубокому хранилищу во время запроса. Это также означает, что у вас должно быть достаточно дискового пространства как в глубоком хранилище, так и во всех исторических процессах.

Глубокое хранение — важная часть гибкой, отказоустойчивой конструкции Druid. Druid может загрузиться из глубокого хранилища, даже если каждый отдельный сервер данных будет потерян и повторно включен.

2. Хранение метаданных. Хранилище метаданных содержит различные общие системные метаданные, такие как информация об использовании сегментов и информация о задачах. В кластерном развертывании это, как правило, будет традиционная СУБД, такая как PostgreSQL или MySQL. При развертывании на одном сервере это обычно локально хранимая база данных Apache Derby.

3. «Смотритель». Используется для обнаружения внутренней службы, координации и избрания лидера.

На рис. 29 показана схема взаимодействия запросов и данных с использованием вышеописанной организации серверов — мастер / запрос / данные (Master/Query/Data) [27].

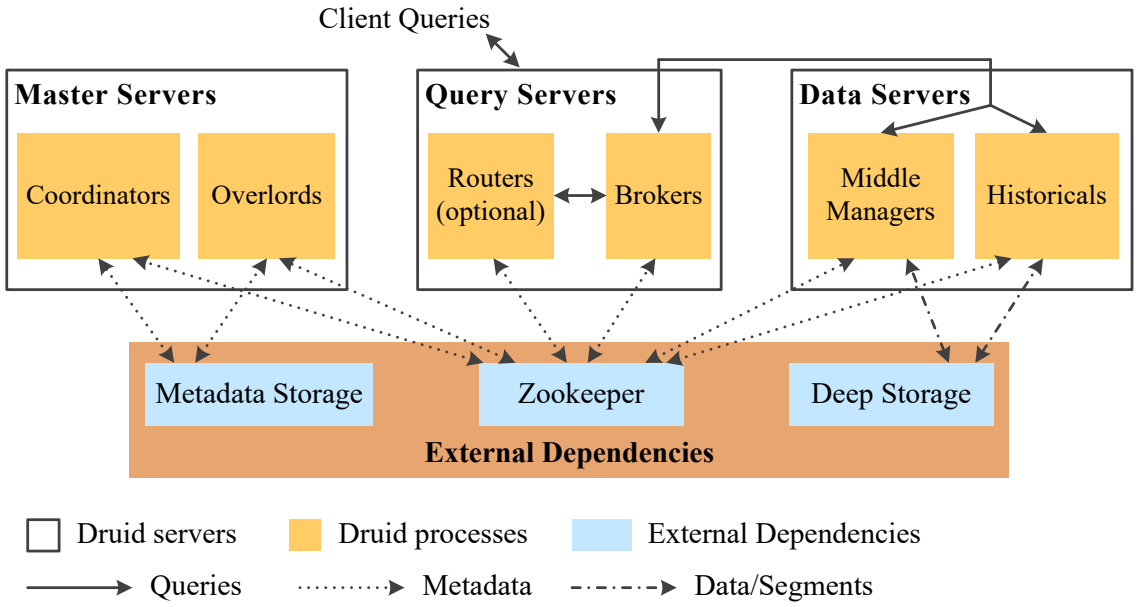


Рис. 29. Архитектура Apache Druid

3.3. Характеристика и примеры применения современных нереляционных СУБД

Свод основных характеристик современных нереляционных СУБД (включая модель данных, поддерживаемый язык, способ распространения, сферу применения и примеры реализации) показан в табл. 17.

Таблица 17

Основные характеристики современных нереляционных СУБД

Название	Используется в системах	Модель данных	Особенности системы	Языки	Сфера применения
Project Voldemort	LinkedIn	Ключ-значение	СУБД с открытым исходным кодом	Java	Территориально распределенные предприятия (имеющие несколько филиалов) с большим количеством серверов (10 и более)
DynamoDB	Nike, Samsung, GE Aviation, Netflix, CapitalOne, Lyft и др.	Ключ-значение	Облачный сервис с оплатой за требуемую производительность (почасовая оплата за пропускную способность)	Java, .NET, PHP, Python и др.	Как отдельные подразделения или производственные элементы, так и целые предприятия, в том числе территориально распределенные
Redis	Verizon, Vodafone, Atlassian, Trip Advisor, Jet.com, Nokia, Instagram, HTC и др.	Ключ-значение	СУБД с открытым исходным кодом	C, C#, C++, Java, PHP Prolog, Visual Basic и др.	Крупные предприятия, в том числе территориально распределенные. Однако, ввиду ограниченности объема хранимых данных объемом оперативной памяти сервера, хранение действительно больших данных сильно увеличивает стоимость использования СУБД
Riak KV, Riak TS	Bump, Rovio (angry birds), Qeep, BestBuy.com и др. [10]	Ключ-значение	СУБД с открытым исходным кодом	Erlang, C, C#, C++, Java, PHP и др.	Крупные предприятия с более или менее сложившейся структурой и имеющие несколько десятков узлов

Продолжение табл. 17

Название	Используется в системах	Модель данных	Особенности системы	Языки	Сфера применения
Aerospike	Nielsen, Williams-Sonoma, Inmobi, AppNexus и др.	Ключ-значение	СУБД с открытым исходным кодом	.NET, C, C#, C++, Erlang, Go, Java и др.	Крупные и средние предприятия, в том числе территориально распределенные
MongoDB	Fordes, The New York Times, Adobe, Amadeus, AstraZeneca, Barclays и др.	Документная система	СУБД с открытым исходным кодом	C, C#, C++, Erlang, PHP, Delphi, Perl Java и др.	Как крупные предприятия, так и небольшие проекты
CouchDB		Документная система	СУБД с открытым исходным кодом	Erlang, Java, .NET, Python, PHP, Ruby и др.	Крупные предприятия, в том числе территориально распределенные
Couchbase		Ключ-значение; документная система	СУБД с открытым исходным кодом	.NET, C, Clojure, ColdFusion, Erlang, Go, Java, PHP и др.	Территориально распределенные предприятия
HBase	Facebook Massanger, Twitter	BigTable	Система с открытым исходным кодом	C, C#, C++, Groovy, Java, PHP, Python, Scala	Территориально распределенные предприятия, социальные сети
Cassandra	Facebook	BigTable	Система с открытым исходным кодом	C#, C++, Clojure, Erlang, Go, Haskell, Java и др.	Крупные непромышленные проекты

Окончание табл. 17

Название	Используется в системах	Модель данных	Особенности системы	Языки	Сфера применения
Adabas	РЖД, Аэрофлот, Администрация Перзидента РФ	NF2	Коммерческая СУБД	Natural	Как отдельные подразделения или производственные элементы, так и целые предприятия, в том числе территориально распределенные
RavenDB	Toyota, Medicaid, Nomura, RMS Automotive	Документная система	Система с открытым исходным кодом	.NET, C#, Go, Java, JavaScript, Python, Ruby	Крупные предприятия, в том числе территориально распределенные

Выводы по главе 3

Существуют следующие основные классы нереляционных систем: системы «ключ-значение»; документные СУБД; системы типа Google BigTable.

NoSQL-системы типа «ключ-значение» хранят данные (неструктурированные или структурированные) и позволяют иметь доступ к ним при помощи единственного уникального ключа.

Документные СУБД используют ориентированную на документ модель для хранения данных. Они хранят каждую запись и связанные с ней данные в одном документе.

Большая таблица — это тип базы данных, которая хранит данные с использованием модели, ориентированной на столбцы. Семейство столбцов состоит из нескольких строк. Каждая строка может содержать разное количество столбцов. Эти столбцы могут иметь разные имена, типы данных и т. д. Каждый столбец содержится в своей строке и не охватывает все строки, как в реляционной базе данных.

В настоящее время существует несколько десятков нереляционных СУБД, как коммерческих, так и с открытым исходным кодом. Модели некоторых из них (Redis, Voldemort, Aerospike, MongoDB, OrientDB, Apache Cassandra, Apache Druid) были подробно рассмотрены в данной главе.

Нереляционные СУБД могут использоваться для достижения различных целей в проектах разных типов и размеров. Общие характеристики с указанием предпочтительной сферы применения также были представлены в данной главе.

Вопросы для самоконтроля

1. Охарактеризуйте понятия «разделение данных» и «репликация». Опишите основные схемы репликации. Назовите основные классы нереляционных систем.
2. Опишите модель данных «ключ-значение». Каковы обязательные требования к ключу и значению в паре ключ-значение? Каковы основные сферы применения систем типа «ключ-значение»?

3. Опишите документную модель данных. Каковы основные отличия документной модели от реляционной? Что представляет собой хранилище документов? В чем сходство и различие документной модели и модели «ключ-значение»? Назовите основные сферы применения документных моделей данных.
4. Дайте понятие «большой таблицы», приведите альтернативные названия этой модели данных. Охарактеризуйте понятие «пространство ключей». Каковы основные элементы строки большой таблицы? Что они содержат?
5. Выполните сравнительный анализ нереляционных моделей Redis, Voldemort, Aerospike, MongoDB, OrientDB, Apache Cassandra, Apache Druid. В каких случаях каждый вид СУБД предпочтительнее использовать?

Глава 4

Методы обработки больших наборов данных

4.1. Краудсорсинг (Crowdsourcing)

Одним из эффективных методов накопления и систематизации данных является краудсорсинг (crowdsourcing, от crowd — толпа и sourcing — использование ресурсов).

Краудсорсинг приравнивают к новым организационным формам исследовательской деятельности [28, с. 175]. Потребность в данном инструменте связана с необходимостью получить отклик на те или иные процессы, тенденции, мнения в течение ограниченного временного интервала.

Термин «краудсорсинг» был предложен в 2006 г. журналистом Джефом Хоу как модель сотрудничества, при которой решение любых, даже самых сложных, проблем доверяется добровольцам [29].

Согласно Хоу, краудсорсинг представляет собой поручение определенной функции, которая раньше выполнялась конкретными работниками, неопределенному кругу лиц в форме открытого конкурса [30, с. 31]. Дж. Хоу выделил ряд типичных для краудсорсинга задач: нахождение решения методом мозгового штурма, создание коллективной базы знаний, коллективное голосование, краудфандинг, коллективное производство [30, с. 31–32].

Начиная с 2010-х гг. краудсорсинг стал удобным инструментом поиска отклика от множества лиц, заинтересованных в решении той или иной проблемы, получив широкое распро-

странение как в научной среде, так и в бизнес-сообществе. Основная предпосылка исследования с использованием данного метода базируется на многообразии общественных интересов и потребностей, их сходстве и различии. Это осуществляется путем зачастую удаленного взаимодействия, при высказывании мнений как можно большего числа респондентов, для которых решение поставленной задачи нередко является областью специализации.

Широкое применение краудсорсинга как метода исследования стало возможным благодаря, с одной стороны, углублению специализации отдельных исследователей в своих областях, с другой стороны, развитию телекоммуникационных каналов связи, что позволило объединить территориально распределенных исследователей в решении какой-либо задачи на выделенном сетевом ресурсе (краудсорсинговой площадке).

В настоящее время данный инструмент стал средством коллективного решения проблем в различных областях знаний, что позволило уточнить и систематизировать круг задач, наиболее подходящих для решения с применением данного метода.

Для демонстрации возможных сфер применения краудсорсинга целесообразно привести классификатор Г. Сакстона, основанный на четырех критериях: вид услуги или продукта, который отдается на аутсорсинг; функции краудсорсеров; степень сотрудничества краудсорсеров и организация проекта [31, с. 2].

Приведенная классификация Сакстона (табл. 18) группирует задачи, решаемые в рамках типовых реализуемых проектов. Как мы видим, во всех случаях реализации краудсорсинговых проектов имеет место запрос задачи от организаторов проекта, а также открытый сбор мнений от лиц, вовлеченных в решение поставленной задачи.

С развитием взаимодействия между пользователями посредством коммуникаций в рамках глобальной сети Интернет в краудсорсинговые проекты вовлекается все больше пользователей прикладного программного обеспечения. Данная вовлеченность реализуется как в игровой форме — в виде целевых игр (порталы типа Fold.it и EteRNA), так и в форме проектов, интегрирующих координаты краудсорсеров и внешние базы данных, например Яндекс.Пробки [30, с. 32]. Преимущество крауд-

Таблица 18

Классификация проектов краудсорсинга согласно Сакстону

Модель	Описание
Модель посредника	Организаторы проекта выступают в роли посредника между заказчиком и исполнителями-краудсорсерами
Производство медиа гражданами	Создание коллективного медиапродукта
Совместная разработка ПО	Адаптация модели «сайта-посредника» для создания компьютерных программ
Продажа цифровых товаров	Электронная торговая площадка, где краудсорсеры размещают цифровой контент — текст, фотографии, видео, графику и т. д.
Разработка дизайна продуктов	Торговая площадка, где краудсорсеры выставляют свои варианты дизайна; создатели проекта налаживают производство и доставку вещей с выбранным дизайном
Децентрализованное финансирование	Площадка для организации взаимодействия заемщиков и инвесторов
Потребительский отчет	Частный случай модели «производства медиа гражданами»: краудсорсеры оставляют только отзывы о товарах или работниках
База знаний	Ответы на вопросы пользователей, аналитические статьи, инструкции, бизнес-идеи и т. д.
Научно-технический проект	Краудсорсерам предлагается решить головоломку, подобрать подпись к изображениям

сорсинга при этом заключается в том, что отбор респондентов не является случайным: как правило, в сборе мнений по тому или иному вопросу участвуют подготовленные и заинтересованные лица, а их отклик в рамках обсуждения либо в ответ на событие является логически аргументированной реакцией.

Отечественные исследователи предлагают классификацию проектов краудсорсинга исходя из уровня участия различных ресурсов (человеческого капитала либо использования электронных ресурсов при принятии решений). В частности, возможна следующая классификация краудсорсинговых проектов.

Первая группа предполагает использовать на входе только «контент человека» [30, с. 34]. К данной группе отнесены проекты, где краудсорсеры поставляют продукт своего труда: текст, фотографию, рисунок, графику, видео и т. д., при этом внешние

ресурсы не импортируются. Такие проекты охватывают широкий круг задач, связанных с разработкой новых продуктов, анализом разрозненной и не структурированной информации и приданием ей систематичности и искомых свойств. К кругу данных проектов можно отнести контент-проекты, направленные на сбор и представление информации путем аккумулирования ресурсов так называемого «коллективного разума» участников различных площадок (порталы для генерации идей и разработки программного обеспечения, базы знаний типа Википедии, базы резюме и вакансий: wikipedia.org, Ответы@mail.ru (otvet.mail.ru) и ряд других [30, с. 35].

Вторая группа объединяет на входе «контент человека» и внешние ресурсы [30, с. 35]. Эти проекты затрагивают привлечение человеческого ресурса для решения поставленной задачи с учетом уже используемых данных — определенных потребительских предпочтений, сформированных моделей, шаблонов и форм. Роль участников проекта сводится к приданию конечному продукту специфических черт и отличительных особенностей.

Внешними ресурсами могут быть не только материалы, но и базы данных. Фактически участники краудсорсиногового проекта занимаются доработкой уже готового продукта и адаптацией его для конечного потребителя на конкретных специфических рынках [30, с. 35–36].

В третью группу попадают проекты, в которых мобильные устройства краудсорсеров поставляют автоматически генерируемый «технический контент» — координаты или показания датчиков, который может дополнять традиционный «контент человека» [30, с. 36]. Проекты позволяют принимать решения с учетом изменившейся обстановки при условии использования технологий массовой обработки данных. В качестве примера можно привести сервис «Делимобиль», использующий определение местоположения краудсорсера. Водители, желающие подработать, посылают в центр обработки данных предполагаемый маршрут, а пассажиры — свое местоположение и место назначения.

Четвертая группа представляет собой на входе «технический контент» и внешние ресурсы и (иногда) «контент человека»

[30, с. 37]. В эту группу включены проекты с импортом внешних ресурсов, в которых краудсорсеры поставляют «технический контент», а иногда и «цифровой контент». Типичный пример — российский проект Яндекс.Пробки, который интегрирует GPS-координаты краудсорсеров-автомобилистов, данные транспортных организаций и наземных видеокамер, а также карты, полученные путем спутникового мониторинга. Сервис позволяет отслеживать загруженность автодорог, прокладывать оптимальный маршрут с учетом дорожных пробок и осуществлять краткосрочный прогноз дорожного трафика [30, с. 37].

Оценка результатов, полученных методом краутсорсинга, находится в родстве с контент-анализом, т. е. предполагает, во-первых, выделение ключевых признаков (или ключевых слов), характеризующих данный объект, во-вторых, определение градаций (состояний), в которых может пребывать объект, в-третьих, создание какого-либо конечного продукта или получение иного результата, характеризующего данный объект в целом.

Возможный результат применения краудсорсинга многообразен, как и количество областей его применения.

В общем, можно охарактеризовать получаемые результаты в ходе реализации краудсорсинговых проектов как выявление узких мест в тех или иных предложенных для публичного обсуждения материалах.

При реализации проектов, направленных на получение отклика от потребителей об удобстве использования того или иного сервиса, производитель сервиса получает систематизированный набор потребительских предпочтений. Дополнительным преимуществом является возможность дифференцированного анализа потребительских предпочтений, собранных в рамках краудсорсингового проекта, в их стратификации по социально-возрастной группе участников. Организатор проекта с применением инструментария краудсорсинга получает отклик об удобстве разрабатываемого прототипа от заинтересованной аудитории.

Преимущества краудсорсинга заключаются в возможности создания с его помощью собирательного образа исследуемого объекта, применяя знания и опыт множества людей, вовлеченных в данный процесс. Нередко краудсорсинговые площадки стано-

вятся центром притяжения экспертного сообщества, способного генерировать множество вариантов решения той или иной проблемы с позиции специфики деятельности. В качестве экспертов могут быть привлечены лидеры мнений в регионе, в области специализации, а также из-за рубежа.

Недостатки: влияние шумов, создаваемых фейковыми аккаунтами и «не профи».

Метод краудсорсинга позволяет объединить специалистов и накопленные ими знания вокруг решения конкретной проблемы и в короткий срок мобилизовать интеллектуальный ресурс сообщества. Современный интернет-инструментарий способствует мобилизации интеллектуального ресурса, сильно распределенного территориально.

4.2. A/B-тестирование

Иным по сути методом работы с информацией является A/B-тестирование, представляющее собой набор из двух альтернатив и их демонстрацию большому количеству пользователей.

Цель A/B-тестирования заключается в поиске наиболее предпочтительного варианта из двух альтернативных (взаимоисключающих).

A/B-тестирование представляет собой метод, позволяющий найти более эффективное решение за счет сбора обратной связи одновременно от большого числа респондентов (пользователей тестируемой системы).

В качестве материала для применения A/B-тестирования можно использовать различные существующие IT-инструменты, рассмотренные в табл. 19.

Исходным материалом для проведения A/B-тестирования является некая альтернатива, предусматривающая наилучший отклик со стороны пользователей. В ходе проведения A/B-тестирования разработчик решает ряд задач, состоящих в получении ответов на некоторые вопросы:

1. Какое визуальное оформление контента наиболее привлекательно для пользователей?

Таблица 19

Особенности применения A/B-тестирования при использовании различных IT-инструментов

Инструмент	Оцениваемый параметр
Лендинги и отдельные посадочные страницы, веб-сайты	Формат посадочной страницы (удобство использования одного из двух вариантов лендингов)
Письма в email-рассылках, посты в социальных сетях	Оценка отклика на один из двух форматов информационного сообщения

2. Каким наилучшим образом можно разместить контент на лендинге для наибольшего отклика пользователей?

3. Какой текст поста получает наибольший отклик от пользователей?

Чтобы получить от A/B-тестирования положительный эффект, необходимо выполнить ряд требований:

1) тестирование альтернатив только в рамках одного изменения. Подача контента двум различным тестовым группам (одной группе — текущий вариант, другой — планируемый вариант) позволит получить отклик об удобстве предлагаемых изменений, а также ускорить адаптацию к данным изменениям;

2) одновременность проведения тестирования в различных аудиториях. Это позволит получать отклик от двух тестовых групп одновременно, без поправки на временной интервал, а также оценить реакцию на точечное изменение одного тестируемого свойства.

К преимуществам данного метода можно отнести достаточно высокую степень объективности оценок пользователей, построенную на визуальных предпочтениях наибольшей доли аудитории. В этой связи A/B-тест представляет собой аналог опроса пользователей посредством информационного контента, только вместо прямых вопросов, предполагающих альтернативные варианты ответа, пользователю предлагается отклик на тот или иной формат контента.

Недостатком метода является длительность разработки инструментов оценки — внедрение тех или иных точечных изменений предполагает трудозатраты команды разработчиков на внедрение инструментов и запуск их в действующей информационной системе.

Также к недостаткам A/B-тестирования можно отнести необходимость последовательного, а не параллельного тестирования доработок, в противном случае у исследователя отсутствует возможность оценить, какая из доработок оказала то или иное влияние.

В целом же А/Б-тест представляет собой эффективный инструмент интернет-маркетинга для повышения лояльности аудитории.

4.3. Прогнозная аналитика

Предсказательная (прогнозная) аналитика (предиктивная аналитика) представляет собой метод анализа данных, позволяющий экстраполировать ретроспективу развития той или иной системы на будущие периоды.

Под прогнозированием следует понимать научное выявление вероятных путей и результатов предстоящего развития явлений и процессов, оценку показателей, характеризующих эти явления и процессы для более или менее отдаленного уровня. Прогнозирование — это научная деятельность, направленная на выявление и изучение возможных альтернатив будущего развития и структуры его вероятных траекторий. Каждая альтернативная траектория развития связывается с наличием комплекса внешних относительно исследуемой системы или явления условий.

Применение методов прогнозирования способствует определению изменчивости анализируемого объекта во времени, а также выработке мер для ответной реакции на данные изменения со стороны субъекта прогнозирования — исследователя, пользователя данных.

Использование прогнозной аналитики неразрывно связано с определением временного интервала, на который целесообразно распространить выявленные тенденции. Данный временной интервал в теории прогнозирования называется периодом упреждения — от момента, для которого имеются последние статистические данные об изучаемом объекте, до момента, к которому относится прогноз.

Предсказательная аналитика широко использует статистический инструментарий, а также инструменты теории игр, анализирующие текущие и ретроспективные факты для формирования предсказаний о предстоящих событиях.

Статистическое описание изменений некоторых величин во времени осуществляется при помощи динамических (временных) рядов, уровни которых формируются под влиянием совокупностей факторов. При этом изменение влияющих факторов приводит к более или менее интенсивной смене значения измеряемого параметра динамического ряда. Статистика выделяет четыре класса компонент, формирующих значение переменной временного ряда: наличие тренда во временных данных, цикличности данных, сезонности и случайной составляющей. Задача исследователя сводится к выделению каждой из данных компонент и измерению силы их воздействия.

При анализе и выделении трендовой составляющей зачастую используется построение регрессионной зависимости рассматриваемого вариационного ряда от фактора времени. Измерение отклонений от предсказанного значения временного ряда в уравнении регрессии интерпретируется как оценка влияния сезонности, цикличности, а также случайной составляющей.

Для оценки временного тренда, как правило, применяются скользящие средние, а также средние темпы роста, представляющие собой среднее геометрическое значение роста анализируемого показателя за рассматриваемый временной интервал.

Прогнозная аналитика позволяет выделить тенденции развития интересующих систем или объектов со статистически определенным уровнем достоверности. Процесс выявления тенденций базируется на методах описанного выше регрессионного анализа, а прогнозирование с его помощью — на экстраполяции данных о выявленных закономерностях на предстоящие периоды.

Экстраполяция тренда представляет собой процесс продления выявленных тенденций, наблюдавшихся в прошлом. Экстраполяция тенденций в силу своей простоты является широко распространенным способом прогнозирования на непродолжительные временные интервалы, в течение которых исследователь не ожидает существенного изменения влияющих факторов.

Ряд допущений делают экстраполяцию трендов способом прогнозирования, ограничивающим его функциональные возможности:

1) развитие явления может быть с достаточным основанием охарактеризовано плавной (эволюторной) траекторией — трендом;

2) общие условия, определяющие тенденцию развития в прошлом, не претерпят существенных изменений в будущем.

Данные предпосылки означают, что экстраполяция определяет общее описание направления будущего развития событий, тем не менее в ряде ситуаций способное достоверно спрогнозировать будущее течение процесса. Причины отклонений фактического развития тренда от прогнозируемого по методу экстраполяции являются, как правило, следствием несоответствия фактических условий развития тем предположениям, которые были первоначально заложены в прогноз.

При определении прогнозных значений анализируемого параметра целесообразно допускать возможность их отклонений от прогноза в пределах величины доверительного интервала, так называемой «вилки» верхнего и нижнего допустимого значения прогноза. Построение доверительного интервала прогноза основывается на статистических методах исследования случайных величин.

Прогнозная аналитика нашла широкое применение в актуарных расчетах, страховании, розничной торговле, здравоохранении и множестве других областей. Одним из направлений является кредитный скоринг, т. е. моделирование склонности потенциальных заемщиков к тому или иному стилю поведения с точки зрения прогноза своевременности выплат по кредитам.

К преимуществам прогнозной аналитики относятся:

1) доступность и распространенность инструментария проведения исследований. Используемый статистический инструментарий (скользящие средние, взвешенные скользящие средние, темпы прироста — среднегодовые, среднемесячные) позволяет провести экспресс-прогнозрование в случае выделения относительно стабильной тенденции;

2) возможность осуществления прогнозирования без применения специального программного обеспечения.

Недостатки прогнозной аналитики:

1) в ряде случаев невозможность учесть множественные параметры и предпосылки для нелинейного тренда;

2) слабый учет качественных сдвигов, возникающих после достижения системой точек бифуркации, так как они построены на количественных, вероятностных методах;

3) исходя из двух предыдущих особенностей целесообразно использовать методы прогнозной аналитики только на коротких временных интервалах.

4.4. Самообучающиеся системы (искусственный интеллект)

В последние годы набирает влияние новый тренд — внедрение в производственные и информационные системы искусственного интеллекта (ИИ). Данный тренд задан в связи с бурным развитием визуального контента и средств обработки массивов данных. Актуальность метода заключается в необходимости экономии ресурсов при организации деятельности крупных промышленных систем за счет апробации результатов на математической модели, с учетом вероятных направлений развития системы и ее реакции на различные внешние воздействия.

Джон Маккарти, автор термина «искусственный интеллект», определил интеллектуальную функцию как вычислительную составляющую способности достигать целей. Само определение искусственного интеллекта Маккарти объяснил как науку и технологию создания интеллектуальных компьютерных программ [32, с. 54].

Для построения адаптивных систем искусственного интеллекта необходима разработка и внедрение таких алгоритмов обработки информации, которые позволяют не только накапливать, хранить и извлекать данные из многообразия доступной информации, но и, адаптируясь к ее качеству, самостоятельно решать вопросы совершенствования технологий ее обработки. Такие алгоритмы называются сложными когнитивными структурами или нейросетевыми структурами. При должном уровне развития и настройки они способны самостоятельно создавать

инструменты для накопления, систематизации, фильтрации и агрегации разнородных данных, характеризующихся неясной и нечеткой структурой. Задача исследователя при этом заключается в создании такого математического аппарата, который обеспечивает возможность воспринимать и распознавать разрозненные данные, в частности, нечетких моделей (fuzzy models), «мягких» вычислений (soft computing), а также эволюционного моделирования и генетических алгоритмов, моделирования рассуждений и неклассической логики, моделирования образного мышления и когнитивной графики, нейронных сетей, генерации и распознавания текста, обработки речи [33, с. 2].

Главная характеристика искусственного интеллекта — выявление обобщающих черт множества объектов, хранимых и обрабатываемых на основе примененных к ним алгоритмов.

Искусственный интеллект зарекомендовал себя как более действенный и эффективный способ в вопросах точной и достоверной оценки и диагностики (оценка финансовых рисков, природно-климатических явлений, диагностика заболеваний) [34, с. 7].

Одной из задач искусственного интеллекта является обработка образов. Данная деятельность предполагает установление визуально-логической связи между имеющимся в наличии электронным образом некоего объекта и оценкой свойств данного объекта. При этом широко используется способность искусственного интеллекта к обучению, к накоплению, хранению и извлечению больших массивов информации, поиску закономерностей в них. Если между входными и выходными данными существует какая-то связь, даже не обнаруживаемая традиционными корреляционными методами, то искусственный интеллект способен автоматически настроиться на нее с заданной степенью точности. Кроме того, он позволяет оценивать сравнительную важность различных видов входной информации [32, с. 55].

При реализации проектов искусственного интеллекта ключевое значение имеет создание модели, обладающей множеством степеней свободы с возможностью самостоятельно эволюционировать в зависимости от цели ее использования.

Основной принцип ИИ — машинное обучение, т. е. создание такого алгоритма, который способен проанализировать боль-

шой объем данных, найти взаимосвязь полученных результатов, построить предиктивные и регрессионные модели [35, с. 91].

Конечной целью применения искусственного интеллекта является экономия ресурсов, направляемых на производство сложных вычислений и построение математических моделей развития. Те задачи, которые ранее решались с привлечением существенных интеллектуальных, трудовых ресурсов в течение длительного периода времени, за счет использования вычислительных мощностей под управлением искусственного интеллекта способны существенно ускорить процесс вывода на рынок новых продуктов, улавливать изменения в трендах, а также производить множество рутинных операций с заданной точностью.

Круг задач, решаемых при помощи искусственного интеллекта, обширен. Он затрагивает задачи анализа свойств и построения прогнозов развития сложных систем в тех случаях, когда взаимосвязи между элементами системы неочевидны и слабо идентифицируются методами корреляционно-регрессионного анализа либо иными методами выявления взаимосвязей и прогнозирования.

Среди основных направлений применения искусственного интеллекта в маркетинге выделяют веб-дизайн, контекстную рекламу, оценку эффективности проведенных рекламных кампаний, поиск по фотографиям, получение сведений рекламодателями для предоставления новостей или рекламной информации [32, с. 55].

Существенное значение данный метод приобрел в финансовой сфере. В исследовании рейтингового агентства «Эксперт РА» «Искусственный интеллект в банковском секторе» говорится о самых перспективных направлениях внедрения ИИ. Тройку лидеров составляют: кредитный скоринг, распознавание мошеннических транзакций и взыскание задолженности [36, с. 28].

Для обеспечения реализации проектов искусственного интеллекта в банковской сфере используется ряд платформ. Например, «Ginimachine» — платформа для оценки кредитоспособности заемщика, построенная на базе технологий машинного обучения. Применение искусственного интеллекта при оценке заемщика банком позволяет повысить доходность кредитного портфеля банка с одновременным снижением уровня риска не-

платежей по кредитам — производить заблаговременный отсев наиболее неблагонадежной категории заемщиков на основе социальных страт и вероятностных моделей поведения заемщиков, относящихся к данным социальным стратам. В настоящее время алгоритмическая модель не только анализирует параметры для принятия решений о выдаче кредитов, но и способна адаптироваться, т. е. самостоятельно оценить целесообразность использования тех или иных параметров для принятия решения.

Существенная роль отведена банковским моделям, анализирующим склонность к мошенническому поведению заемщиков, в связи с чем в российской и международной практике также протестированы и внедрены модели мониторинга поведения банковских клиентов, распознавания и блокирования сомнительных операций.

Применение искусственного интеллекта также нашло свое отражение при реализации технологий функционирования чат-ботов, т. е. программ, разработанных на основе технологий машинного обучения и нейросетей для достижения человеком определенных целей. Функционал чат-бота позволяет организовать взаимодействие в системе «человек — информационная система» за счет распознавания голосовых команд, а также нечетких команд. Деятельность по разработке чат-бота и наделению его чертами личности требует всесторонней оценки каналов коммуникаций для максимального приближения качества диалогов с чат-ботом к диалогам между людьми. Специалисты, занятые разработкой дизайна личности чат-бота, опираются на знания из различных социальных наук, таких как психология, филология, драматургия. Целью данной синергии является создание способов построения максимально эффективных диалогов человека и машины. Стандартная подготовка к созданию диалогового интерфейса включает формирование личности чат-бота (сравните: образ оратора в античном риторическом каноне), определение цели диалога и особенностей диалога (сравните: этап инвенции в риторике), построение общей логики процесса общения (сравните: выбор риторической стратегии и речевых тактик), написание детального сценария диалога. При создании «одушевленного» бота общие рекомендации практиков, специализирующихся на продвигающей коммуникации, сводятся к необходимости об-

ращаться к адресату от имени бренда (например, телеканала), требованию приватного общения, лаконичности диалога, требованию выдавать советы адресату частями, соблюдать рамки приличия в шутках, а также к требованию не пропускать реакцию персонажа (чат-бота) на слова и действия человека [37, с. 49].

В ходе разработки инструментов искусственного интеллекта создается продукт, способный хранить и накапливать данные, дополнять их, систематизировать, а также производить вычисления и выполнять операции с различной степенью сложности, зависящей от целей использования того или иного алгоритма, а также от уровня его обученности, т. е. широты используемых данных.

Постепенно искусственный интеллект и системы, построенные при его участии, получают все большее распространение. Преимущества метода заключаются в существенном ускорении процессов, для которых данные алгоритмы разрабатываются. Самообучающаяся система, способная производить вычисления и представлять их результат, может высвободить человеческий ресурс за счет принятия на себя рутинных циклических задач. Помимо этого, такая система склонна к гораздо меньшему количеству ошибок, вызванных неверным использованием данных, неверными вычислениями. К преимуществам также можно отнести скорость производимых вычислений.

Недостатком применения искусственного интеллекта является необходимость существенных трудозатрат на стадии проектирования и тестирования алгоритмов. Авторам алгоритмов за счет дробления анализируемых объектов на множество составных частей и описания как свойств этих частей, так и свойств целостных объектов необходимо научить искусственный интеллект не ошибаться. Данная задача представляет собой итерационный процесс, состоящий из множества тестирований и ручных проверок на стадии внедрения и пилотирования системы.

Одной из задач развития систем искусственного интеллекта является наделение их механизмом поиска самостоятельных решений в нестандартных ситуациях, т. е. в ситуациях, отклоняющихся от описанных в исходном коде алгоритма. Данная проблема связана с наделением алгоритмов алгоритмической способностью распознавать нестандартные ситуации и адекватно

реагировать на них. Сложность заключается в сокращении относительной частоты проявления нестандартных ситуаций в зависимости от объема наполнения алгоритма искусственного интеллекта данными и знаниями.

В настоящий момент внедрение систем искусственного интеллекта находится на стадии зарождения, но успех подобных программ, несмотря на имеющиеся в технологии недостатки (определение оптимальной архитектуры моделируемых искусственных нейронных сетей, потребность в дорогой компьютерной технике, способной справиться с трудоемкими вычислительными алгоритмами, отсутствие необходимых информационных библиотек и баз данных), бесспорен. Электронно-вычислительные системы, получившие поддержку ИИ, показывают более точные, достоверные и эффективные результаты.

4.5. Сетевой анализ

Еще одним методом, применяемым при обработке больших массивов данных, является сетевой анализ. Анализ сетей позволяет использовать различные подходы к исследованию данных и выявлению взаимозависимостей: статистические, кибернетические, системные, имитационные.

Применение сетевого анализа позволяет структурировать связи, установленные взаимодействия между различными социальными единицами: людьми и организациями. Для описания взаимодействий между элементами системы посредством установления социальных связей между ними используется понятие «социального графа», т. е. способа визуализации многообразия исследуемых взаимозависимостей между множеством элементов в пространстве, в том числе в виртуальной среде.

При взаимодействии в рамках построенной сети все ее элементы обмениваются ресурсами и информацией в пределах выстроенных взаимосвязей. Установление порядка взаимосвязей производится на основании API.

Социальный граф предполагается хранить в виде списка смежности, где в качестве ключа используется идентификатор пользователя, а список содержит идентификаторы тех, на кого

подписан текущий пользователь [38, с. 71]. Вершины в графе представляют объекты (или субъекты, «акторы»), а ребра — связи между ними. При этом наличие связи может характеризовать близость (похожесть) между объектами, различие (расстояние) между ними или факт (интенсивность) их взаимодействия [39, с. 57–58].

Применение графов нашло свое отражение в описании моделей распределения и движения тех или иных ресурсов между акторами для их последующей оптимизации, т. е. ускорения оборачиваемости ресурсов, ускорения прохождения потоков информации. Данная деятельность получила название сетевого планирования.

Во взвешенных графах каждому ребру приписано некоторое число (вес), характеризующее меру близости или расстояния (степень похожести или различия) между объектами или интенсивность воздействия (влияния) одного объекта на другой, объем (величину) материального, информационного или финансового потока между ними и т. п. [39, с. 62].

В ходе построения сетевых взаимосвязей ключевую роль играет определение мер центральности тех или иных факторов взаимодействий. Центральное расположение является отражением меры того, насколько данные факторы, т. е. узлы системы, способны концентрировать ресурсы и потоки, перераспределяемые и направляемые системой. Наиболее известны и распространены следующие показатели [40; 41]:

– центральность по степени связности (degree centrality);
– центральность по близости (closeness centrality);
– центральность по посредничеству (betweenness centrality);
– центральность по влиятельности (eigenvector centrality).

Рассматривая каждый из данных показателей, можно определить ключевые параметры, оцениваемые для каждой сетевой модели (табл. 20).

Концепция центральности широко используется в социальных и поведенческих науках, а также в области политических наук, управления, социальных сетей в Интернете, экономики, биологии и т. д.

Таблица 20

Описание мер центральности

Показатель	Сущность центральности	Оцениваемые параметры
Центральность по степени связности (degree centrality — DC)	Обладание наибольшим количеством связей (отношений) с другими факторами	DC находится в пределах от 0 до 1 и тем выше, чем большее число связей установлено у конкретного фактора взаимодействия с другими факторами в графе. Фактор с высоким значением DC взаимодействует с большим числом других узлов в сети и тем самым может получить доступ к большому объему других узлов и ресурсов, оказывать влияние на них
Центральность по близости (closeness centrality — CC)	Близость расположения определенного фактора от множества других в рамках сети	Характеризует центральность нахождения вершины на основе расстояния от центрального участника до других вершин графа, т. е. определяет, насколько данный узел близок ко всем остальным узлам в сети
Центральность по посредничеству (betweenness centrality — BC)	Ценность данного фактора как посредника	Подход заключается в нахождении доли самых коротких (кратчайших) путей, соединяющих все пары вершин, которые проходят через данную вершину
Центральность по влиятельности (eigenvector centrality — EC)	EC является вариацией DC, в которой ведущая роль отдана количеству узлов, прилегающих к данному. По существу, это мера оценки популярности того или иного узла	Рассматривается качество влияния связанных узлов путем измерения оценок влияния узла на одном конце связи и на другом конце. Степень влиятельности одного участника взаимодействия определяется уровнем влиятельности его партнеров, что делает EC показателем оценки власти того или иного участника взаимоотношений

При использовании сетевого анализа в исследовании взаимосвязей на рынке межбанковского кредитования и платежных

систем производится количественная оценка значимости тех или иных факторов при исследовании элементов финансовых взаимодействий и накопленного риска. Оценки влияния и меры центральности используются при определении ключевых игроков на рынке товаров, услуг, капитала, а также при исследовании политических систем. Количество областей применения велико. Одним из наиболее ярких примеров является использование данных социальных сетей, которые позволяют сформировать и в определенной степени формализовать анализируемые взаимозависимости между различными факторами.

Результатом применения сетевого моделирования является модель распределения ресурсов некой системы: общества, государства, компании. Модель позволяет оперировать множеством взаимосвязей между отдельными объектами и динамически оценивать значимость каждого из факторов сетевого взаимодействия.

Среди преимуществ данного метода следует указать способность к визуализации взаимосвязей между объектами в рамках одной системы. Факторы, входящие в систему, обладая различными параметрами, характеризующими их центральное либо отдаленное положение, обеспечивают влияние на смежные элементы. Данное влияние необходимо оценивать динамически, по мере изменения расстановки сил в рамках системы.

К недостатком метода относится трудоемкость установления взаимосвязей между элементами сложной системы. При росте количества элементов системы возникает необходимость дополнения данных об изменениях порядка и силы взаимосвязи. Заранее неизвестно, как на общее состояние системы повлияет включение в нее дополнительных факторов и насколько подвержено изменению текущее состояние факторов, уже включенных в модель.

Рассмотренные в данной главе методы работы с разнородными данными можно охарактеризовать как систему оценки взаимосвязей и взаимовлияния между множеством уровней развития различных систем, а количественные числовые оценки данных взаимосвязей — как меру значимости тех или иных элементов.

Выводы по главе 4

В данной главе были рассмотрены основные средства обработки и анализа больших данных (краудсорсинг, A/B-тестирование, прогнозная аналитика, самообучающиеся системы, сетевой анализ).

Краудсорсинг является удобным инструментом поиска отклика от множества лиц, заинтересованных в решении той или иной проблемы.

A/B-тестирование представляет собой набор из двух альтернатив и их демонстрацию большому количеству пользователей с целью сбора обратной связи от пользователей, отдающих предпочтение той или иной альтернативе.

Прогнозная аналитика — метод анализа данных, позволяющий экстраполировать ретроспективу развития на будущие периоды.

Самообучающиеся системы способны вырабатывать решения исходя из результатов анализа эффективности решений, принятых в прошлом.

Сетевой анализ с целью выявления взаимозависимостей в исследуемых данных комбинирует системные, статистические, кибернетические, имитационные и прочие методы.

Вопросы для самоконтроля

1. Опишите основные принципы и сферы применения краудсорсинга.
2. Опишите основные принципы и сферы применения A/B-тестирования.
3. Опишите основные принципы и сферы применения прогнозной аналитики.
4. Опишите основные принципы и сферы применения самообучающихся систем.
5. Опишите основные принципы и сферы применения сетевого анализа.

Глава 5

СУБД Adabas: функциональные возможности для анализа больших наборов данных

Одной из первых реализаций, обладающих инструментарием для реализации основных моделей данных нереляционных СУБД, можно считать промышленную СУБД Adabas, созданную в 1971 г. Это постреляционная СУБД, поддерживающая различные модели данных, включающая вложенные отношения и иерархические поля. С помощью Adabas можно строить как иерархические, сетевые и реляционные SQL базы данных, так и сложные текстовые информационно-поисковые и интегрированные системы, а также реализовать современные модели нереляционных баз данных.

5.1. Базовая модель и структура данных СУБД Adabas

Структура данных базовой модели

База данных Adabas определяется как совокупность взаимосвязанных файлов и ассоциаций записей файлов.

Файл представляет собой именованную совокупность записей, имеющих одинаковую структуру, компонентами которой являются атрибут, множественный атрибут, простая, составная и повторяющаяся группа. Каждой записи файла назначается системный ключ, представляющий собой число в диапазоне от 1 до $16 \cdot 10^6$, которое однозначно идентифицирует экземпляр записи и определяется в Adabas как внутрисистемный номер.

Связи между файлами (типа 1:1, 1:*M* или *M*:*N*) являются непоименованными парными связями и предназначены для моделирования сетевых и иерархических отношений между объектами предметной области, а также групповых отношений в объектах иерархической структуры.

Ассоциация записей файлов (далее ассоциация) представляет собой совокупность записей файла, обладающих общим свойством. В качестве такого свойства может выступать, в частности, значение некоторого атрибута, которое совместно с его именем определяет однозначно ассоциацию и может использоваться как ключ для доступа к ассоциации записей файла.

Ассоциации в Adabas организуются в БД в виде списков внутрисистемных номеров (ВСН) записей файлов. Если ассоциация образуется на основе равенства значений некоторого атрибута, список ВСН записей, входящих в ассоциацию, называется инвертированным списком, а сам атрибут — поисковым. В том случае, когда ассоциация образуется на основе равенства значений атрибута записей одного файла значению атрибута некоторой записи другого файла, соответствующий список ВСН называется списком связи, а указанные атрибуты файлов — атрибутами связи.

Использование ассоциаций реализует, по существу, разбиение записей файлов на классы, поддерживаемое в Adabas динамически в процессе модификации БД. При этом ассоциации в виде инвертированных списков обеспечивают возможность ускоренной селекции записей файлов по их содержимому, а ассоциации в виде списков связь — возможность поддержания связей между файлами и возможность селекции записей с применением этих связей.

Описание структуры базы данных на языке описания данных (ЯОД) определяется как схема БД. Схема базы данных

включает схемы составляющих ее файлов и схемы связей между ними [42, с. 152].

Как уже отмечалось, к числу компонентов структуры файла относятся атрибут и группа. Атрибут является наименьшей именованной единицей данных, каждый экземпляр которой в записи файла представляется одним или несколькими значениями. Группа также рассматривается в виде атрибута, который называется множественным.

Для атрибута в схеме файла задаются его короткое двухсимвольное имя (определяемое на ЯОД как код имени атрибута), тип и максимальная длина значения; для множественного атрибута дополнительно может быть определено максимальное количество экземпляров его значения в записи.

Тип значения атрибута определяет стандартный (принятый по умолчанию в командах языка манипулирования данными — ЯМД) вид представления значения этого атрибута в прикладной программе. К числу допустимых типов значений относится символьный, упакованный и распакованный десятичный, битовый и двоичный.

Атрибут в схеме файла может быть описан как уникальный, при этом его значения однозначно определяют экземпляры записей в файле БД и могут быть использованы в качестве ключей записей.

Группа представляет собой именованную совокупность атрибутов и, возможно, других групп. Простая группа состоит только из атрибутов, составная группа может содержать как атрибуты, так и простые и составные группы.

Периодической называется группа, которая может иметь в записи несколько экземпляров. Периодические группы в структуре файла не должны быть вложенными, т. е. не должны входить в состав других групп. Простые и составные группы могут быть иерархически подчинены одна другой и входить в состав повторяющихся групп, образуя древовидную структуру.

При описании группы на ЯОД задаются ее имя и двухсимвольный код имени, для повторяющихся групп может быть задано максимальное количество экземпляров этой группы в пределах экземпляра записи.

Использование группы упрощает доступ к входящим в ее состав атрибутам и группам. В команде выборки достаточно указать код имени группы, чтобы получить значения всех входящих в нее атрибутов. В случае периодической группы необходимо также указать номер (индекс) требуемого экземпляра.

Периодическая группа является средством реализации связи типа 1:*M* в пределах записи. Пример использования периодической группы в составе структуры файла СОТРУДНИКИ представлен на рис. 30, 31.

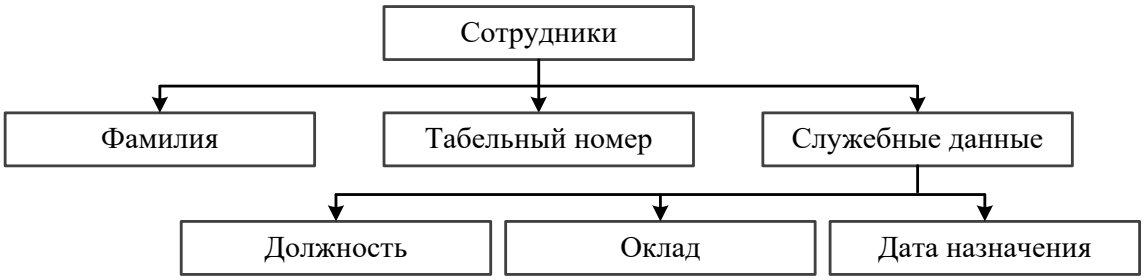


Рис. 30. Структура файла СОТРУДНИКИ
с повторяющейся группой СЛУЖЕБНЫЕ ДАННЫЕ

		Инженер	140	17.08.2005
Николаев	983	Лаборант	90	20.03.2003
		Ст. инженер	160	23.02.2001
		Инженер	140	19.06.2000
Петров	2013	Техник	80	06.04.1997

Рис. 31. Записи файла СОТРУДНИКИ
с повторяющейся группой СЛУЖЕБНЫЕ ДАННЫЕ

Как отмечалось ранее, ассоциации записей файла в базе данных Adabas организуются в виде инвертированных списков по значениям поисковых атрибутов и в виде списков связи — по значениям атрибутов связи.

Поисковый признак для соответствующего атрибута указывается при его описании и структуре файла, а коды имен атрибутов связи — при определении связи двух файлов на ЯОД. Предусматривается также возможность организации ассоциаций по значениям производных атрибутов.

Значение производного атрибута может быть определено в схеме файла как часть значения обычного (непроизводного) атрибута (усеченный производный атрибут) или как конкатенация значений и/или частей значений обычных атрибутов (составной атрибут). Значения производных атрибутов используются только для организации ассоциаций и в записях файлов БД не хранятся [42, с. 154].

Производные атрибуты по своей сущности являются поисковыми и могут использоваться также в качестве атрибутов связи. Ассоциации, организованные по значениям производных атрибутов, расширяют возможности динамической классификации записей файла, позволяя моделировать классификацию объектов предметной области не только по их свойствам, но и по комбинации свойств.

Ассоциации записей файлов БД, реализованные посредством инвертированных списков и списков связи, автоматически создаются и корректируются при вводе, корректировке и удалении записей файлов, содержащих значения поисковых атрибутов, атрибутов связи и значения атрибутов, входящих в состав значений производных атрибутов.

Пример организации и ведения инвертированных списков рассматривается на рис. 32–34. Каждая запись представлена с ее внутрисистемным номером, который не входит в состав структуры записи. Поисковыми в схеме файла КАДРЫ являются атрибуты ГОД-РОЖДЕНИЯ и ПОЛ с кодами имен соответственно GR и PL. Файл КАДРЫ первоначально содержит восемь записей. По значениям поискового атрибута ГОД-РОЖДЕНИЯ для представленных записей организовано три инвертированных списка, по значениям поискового атрибута ПОЛ — два.

Рис. 32. Схема файла

Индексные таблицы, обеспечивающие быстрый доступ к инвертированным спискам по коду имени и значению поискового атрибута, изображены на рис. 33, 34 упрощенно.

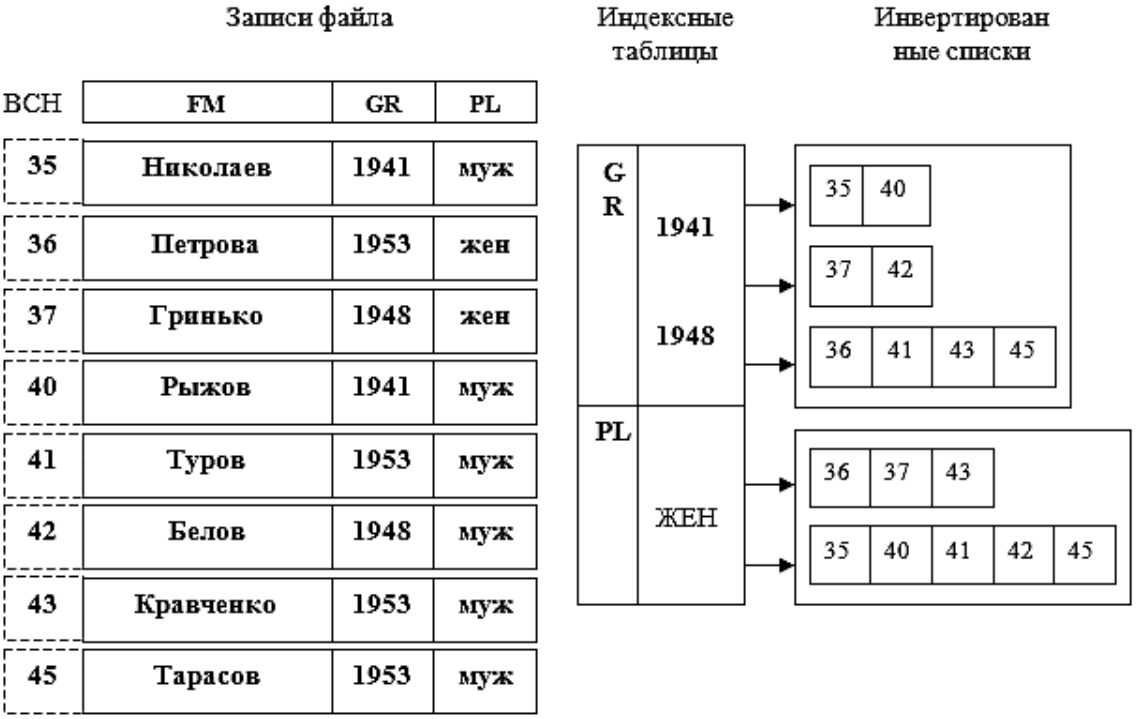


Рис. 33. Записи файла и инвертированные списки до модификации файла

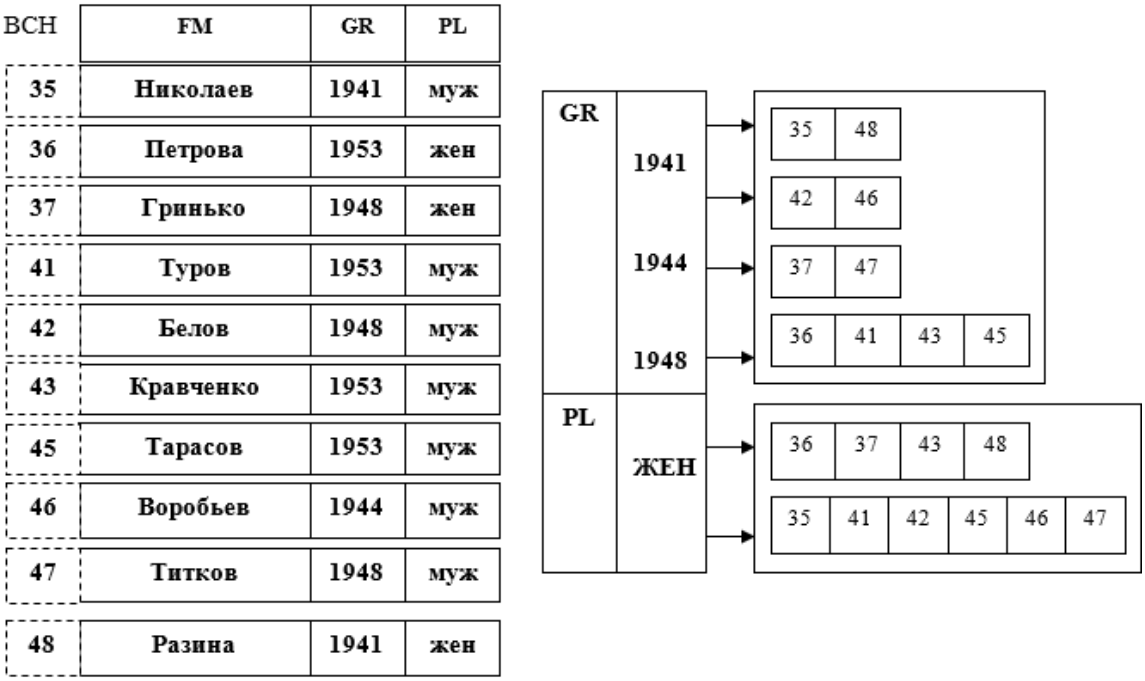


Рис. 34. Записи файла и инвертированные списки после модификации файла

Модификация файла, которая заключалась в удалении записи 40, добавлении записей 46, 47 и 48 и корректировке записи 42 (замена ошибочного значения 1948 на значение 1944), привела к созданию нового инвертированного списка с соответствующей модификацией индексной таблицы и корректировке всех инвертированных списков, кроме одного (рис. 34).

Рассмотрим далее организацию связей между записями файлов БД. Между любыми двумя файлами БД возможна организация одного типа связи, причем каждый файл может быть связан с несколькими другими файлами.

При описании связи на ЯОД указываются номера связываемых файлов и имена атрибутов связи (по одному в каждом файле). Запись любого из этих файлов объявляется связанной с одной или несколькими записями второго файла, если атрибут связи в каждой из этих записей имеет значение, совпадающее со значением атрибута связи в записи первого файла.

Для записи одного файла, связанной с множеством записей другого файла, в БД строится список связи, включающий совокупность ВСН этих записей. Доступ к списку связи осуществляется по ВСН записи первого файла, с которой связаны записи, представленные в списке связи. Первичное создание и удаление списков связи производится в автономном режиме с помощью утилит.

Существенно, что атрибуты связи в каждом из связываемых файлов должны быть поисковыми (объявленными дескрипторами), поскольку формирование списков связи осуществляется с помощью инвертированных списков, которые до связывания должны быть организованы по каждому из этих атрибутов. Для организации связи типа один-к-одному в качестве атрибутов связи в каждом из файлов должны быть объявлены уникальные атрибуты. Связь типа один-ко-многим и типа многие-к-одному будет организована, если только один из атрибутов связи будет уникальным, а второй — простым атрибутом.

Связь типа многие-ко-многим может быть организована при объявлении атрибутом связи в одном файле уникального атрибута, а во втором — множественного атрибута. На атрибуты связи накладывается ряд ограничений, в число которых входят следующие:

– тип и длина значения атрибутов связи в обоих файлах не должны различаться;

– множественным может быть атрибут связи только в одном из файлов;

– атрибут связи не должен входить в повторяющуюся группу;

– производный атрибут, используемый в качестве атрибута связи, не может быть сформирован из атрибутов, входящих в повторяющиеся группы.

Пример организации списков связи для связи типа 1:*M* представлен на рис. 35−37. Файл СТУДЕНТ содержит атрибуты ФАМИЛИЯ (код имени FM) и ГРУППА (код имени GR), а файл КАРТОЧКА — атрибуты КНИГА (код имени KN) и ЧИТАТЕЛЬ (код имени CH). Связывание производится по значениям уникального атрибута ФАМИЛИЯ в первом файле и атрибута ЧИТАТЕЛЬ — во втором. Списки связи формируются путем последовательного просмотра инвертированных списков обоих файлов. При этом для каждого BCH очередного инвертированного списка файла по значению поискового атрибута этого списка (он же является атрибутом связи) происходит обращение к соответствующему инвертированному списку второго файла, который заносится в БД как список связи с ключом, в качестве которого используется исходный BCH записи первого файла. В результате последовательного выполнения указанных операций формируются списки связи для каждого из связываемых файлов, при этом «прямые» списки для первого файла будут «обратными» для второго.

Списки связи хранятся так же, как инвертированные списки, при этом для ускорения доступа к спискам связи создается индекс, отсортированный по значениям BCH, используемым в качестве ключей списков связи.

Связи между записями файлов, соответствующие списку связи с ключом 4, на рис. 35 помечены пунктиром.

Более сложный пример связывания файлов представлен на рис. 38−40. Схема файла СТУДЕНТ включает атрибуты НОМЕР-ЗАЧЕТНОЙ-КНИЖКИ, ФАМИЛИЯ и ГРУППА с кодами имен соответственно NZ, FM и GR, а схема файла СЕМИНАР — атрибуты РУКОВОДИТЕЛЬ, КАФЕДРА и УЧАСТНИК с кодами

имен RK, KF и UK. Атрибутами связи файлов являются атрибуты НОМЕР-ЗАЧЕТНОЙ-КНИЖКИ и УЧАСТНИК.

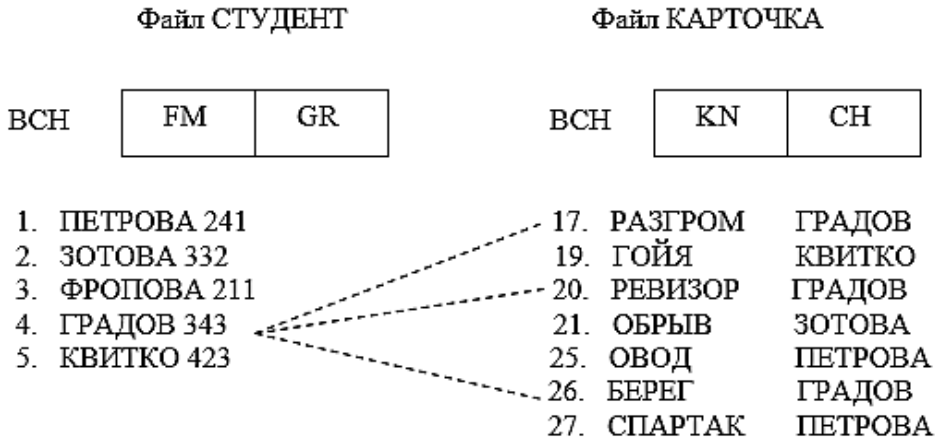


Рис. 35. Записи файлов (один-ко-многим)

Рис. 36. Инвертированные списки с индексными таблицами (один-ко-многим)

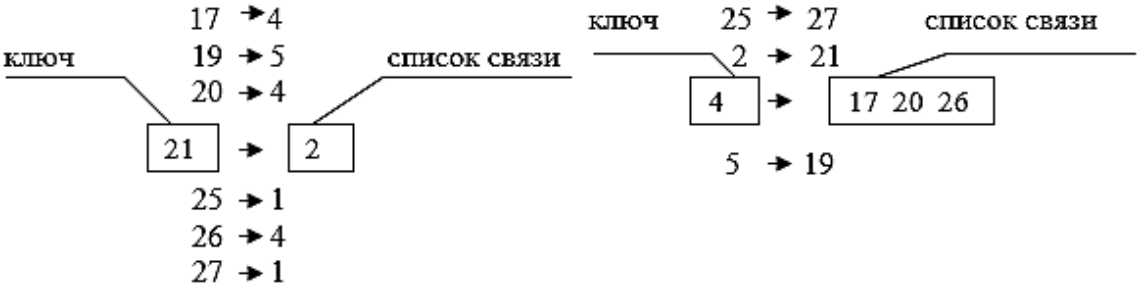


Рис. 37. Списки связи с ключами (один-ко-многим)

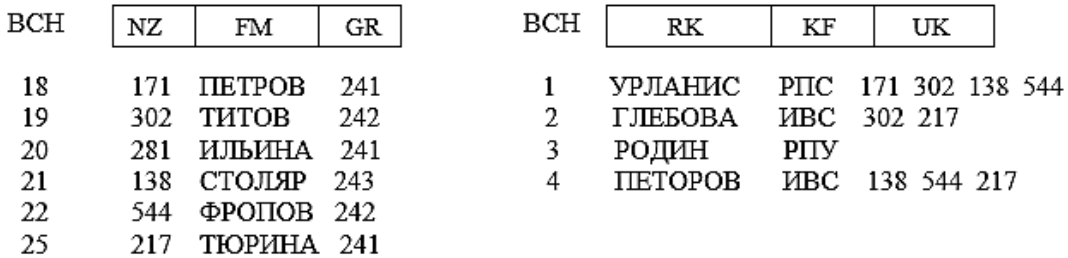


Рис. 38. Записи файлов (многие-ко-многим)

Рис. 39. Инвертированные списки с индексными таблицами (многие-ко-многим)

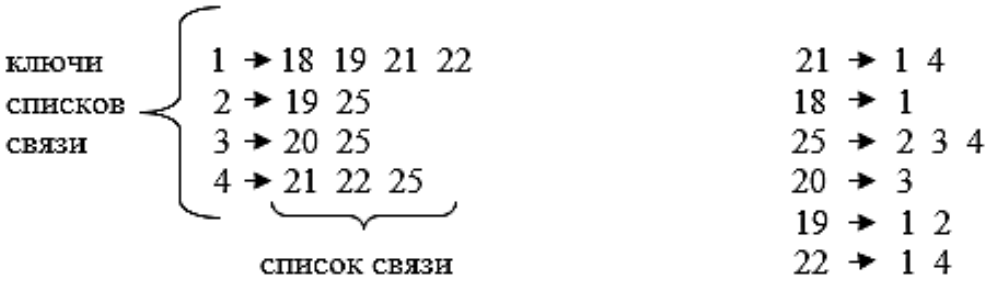


Рис. 40. Списки связи с ключами (многие-ко-многим)

В данном случае реализуется связь типа многие-ко-многим, и алгоритм формирования «прямых» списков второго файла несколько сложнее, так как каждый список связи формируется в результате объединения отдельных одноэлементных «частных» списков.

При организации связи между двумя файлами БД записи считаются равноправными, и связь может использоваться в обоих направлениях.

По способу реализации связь между файлами Adabas может быть отнесена к типу связей, не несущих информации, поскольку все, что семантически связывает записи двух файлов между собой, содержится в значениях соответствующих атрибутов записей независимо от того, созданы или нет списки связи.

Исходя из вышеизложенных принципов организации связей между записями файлов БД системы Adabas работа с логически связанными записями различных файлов возможна только с помощью инвертированных списков, без организации списков связи. Однако такой способ связывания файлов в процессе обращения к БД требует дополнительных временных затрат, поэтому использование списков связи при работе с сетевыми

и иерархическими структурами повышает производительность системы.

Независимая от записей файлов организация ассоциаций в виде инвертированных списков и списков связи в Adabas обеспечивает возможность создания новых ассоциаций и уничтожения старых без перезагрузки файлов БД.

Операции базовой модели данных

Объектами операций модели данных концептуального уровня Adabas являются БД и составляющие ее компоненты.

Операция создания БД заключается в формировании внутримашинного представления схемы, т. е. загрузочной схемы БД. Остальные операции, выполняемые над схемой БД и значениями составляющих БД компонентов, относятся к операциям модификации и доступа к данным.

Операции модификации схемы БД предусматривают:

– добавление схемы нового файла;

– удаление существующей схемы файла;

– включение в структуру существующего файла новых атрибутов, групп и производных атрибутов;

– установление новых связей между файлами;

– удаление существующих связей между файлами;

– определение непоискового атрибута поисковым (объявление поля дескриптором);

– отмену признака поисковости у атрибута (отмена объявления поля дескриптором).

Операции модификации данных, хранящихся в БД, обеспечивают ввод, корректировку и удаление отдельных записей файлов БД. Для выполнения указанных операций в ЯМД концептуального уровня Adabas предусмотрены соответствующие команды. Массовый ввод и обновление записей файлов, а также удаление файлов БД осуществляются утилитами автономного ведения данных. Операции модификации данных включают:

– ввод записей в БД;

– корректировку записей файла;

– удаление записей файла;

– доступ к данным:

а) чтение записей файла;

б) поиск записей файла;

– селекцию по связям записей файла;

– рандомизированный доступ к записям.

Ввод записей в БД. Запись, вводимая в БД программой пользователя, должна содержать не менее одного атрибута. Каждый атрибут в составе записи представляется при вводе кодом имени и значением, длиной и типом значения атрибута.

Вводимой записи автоматически присваивается ВСН, который после окончания операции ввода передается прикладной программе. Предусмотрена также возможность назначения ВСН программой пользователя по известному ему алгоритму, однако назначаемый подобным образом ВСН не должен превышать максимально допустимого числа записей в файле, которое задается параметрически при создании файла.

Если во вводимой записи имеются значения атрибутов связи и поисковых атрибутов, а также значения атрибутов, входящих в состав производных атрибутов, то модификация списков связи и инвертированных списков, т. е. создание новых и корректировка существующих списков, выполняется автоматически.

Все атрибуты, которые не вводятся в составе новой записи и в схеме файла предшествуют последнему вводимому атрибуту, после загрузки получают нулевое значение для числового и значение «пробел» — для символьного типа значения. Это означает, что в качестве признаков «неопределенных» значений в записях файлов БД системы Adabas используются нуль или пробел в зависимости от типа значения атрибута.

Если атрибут с «неопределенным» значением является поисковым (или атрибутом связи), то при описании его на ЯОД без признака С ПОДАВЛЕНИЕМ его значение (нуль или пробел) используется для модификации инвертированных списков и списков связи. В противном случае, т. е. если поисковый атрибут, значения которого не вводятся, определен С ПОДАВЛЕНИЕМ, модификация инвертированных списков и списков связи при выполнении операций ввода не производится. Это относится также к невводимым поисковым атрибутам, которые в схеме файла находятся за атрибутом, значение которого в составе вводимой записи является последним.

Если в составе записи находится значение уникального атрибута, то при выполнении операции ввода производится проверка на отсутствие этого значения в ранее введенных записях файла.

Корректировка записей файла. Единичная запись как объект корректировки идентифицируется ее внутрисистемным номером. ВСН записи получается в прикладной программе с помощью одной из операций селекции данных.

Логика операции корректировки записей файла основана на сопоставлении значений атрибутов, заданных программой пользователя, и значений этих атрибутов в корректируемой записи файла БД.

Корректировка записей файла заключается в обновлении значений атрибутов этих записей. При этом обеспечиваются вставка, замена и исключение значений атрибутов ранее введенных записей.

Если значение обновляемого атрибута задано в прикладной программе, а в корректируемой записи значение этого атрибута отсутствует, производится вставка нового значения атрибута в состав существующей записи файла.

При задании значения атрибута в прикладной программе и отличном от него значении этого атрибута в корректируемой записи существующее значение атрибута записи файла заменяется на новое значение.

Значения атрибута записи удаляются путем замены существующего значения атрибута на «неопределенное» (нулевое — для числовых или пробелы — для символьных атрибутов), которое задается для удаляемого атрибута прикладной программой.

Если в число обновляемых атрибутов при корректировке записей файла входят поисковые атрибуты, производятся необходимые изменения в инвертированных списках и списках связи. Модификация списков связи и инвертированных списков, построенных по значениям производных атрибутов, осуществляется при обновлении значений атрибутов, из которых сформированы значения производных атрибутов.

Удаление записей файла. Идентификация удаляемых из файла записей осуществляется по ВСН этих записей, причем одна операция рассчитана на удаление одной записи.

Занятая удаляемой записью физическая память файла объявляется свободной для дальнейшего использования, а ВСН удаленной записи отмечается как незанятый и в дальнейшем может быть назначен вновь вводимым записям.

Удаление записи сопровождается модификацией соответствующих списков связи и инвертированных списков (см. рис. 32–34).

Доступ к данным. Операции доступа к данным представлены в модели данных концептуального уровня Adabas операциями чтения и поиска записей файлов БД.

Операция чтения обеспечивает селекцию записи по ее позиции в файле или его части и выборку требуемых значений атрибутов, т. е. операция чтения эквивалентна последовательности операций селекции и выборки данных.

Операция поиска обеспечивает селекцию некоторой совокупности записей файла по значениям атрибутов этих записей или с учетом связей между файлами и, при необходимости, выборку значений атрибутов из первой отобранной записи. Частным случаем селекции по значениям атрибутов в Adabas является рандомизированный доступ к записям файлов.

Чтение записей. В зависимости от вида упорядоченности множества записей операция чтения обеспечивает доступ к записям файла:

– по списку ВСН записей;

– в порядке возрастания ВСН записей файла;

– в логической последовательности по значениям заданного поискового атрибута;

– в физической последовательности расположения записей в БД.

При доступе к записям файла по списку ВСН в качестве упорядоченного множества выступает часть записей файла БД, представленная списком ВСН, причем возможно обращение к первой заданной и к следующей записи этого множества. Сам список должен быть отсортирован по возрастанию значений ВСН, он может быть получен с использованием селекции по значениям атрибутов или селекции по связям файлов.

Доступ к записи файла по ВСН осуществляется через системную таблицу, называемую преобразователем адреса. Преоб-

разователь адреса организуется для каждого файла БД и отображает каждый ВСН в относительный номер блока набора данных, в котором размещается запись файла с этим ВСН.

При доступе к записям файла в порядке возрастания их ВСН возможно обращение к записи с минимальным ВСН, к записи с ВСН, ближайшим «большим» указанного, и обращение к записи со следующим (точнее, ближайшим — большим) ВСН независимо от физического размещения записей.

Следует отметить, что наряду со значениями требуемых атрибутов очередной записи файла пользовательской программе становится доступным ВСН этой записи.

При доступе к записям в логической последовательности значений поискового атрибута в качестве упорядоченного множества, в котором производится селекция записей, выступает вся совокупность ассоциаций, построенных по значениям заданного поискового атрибута файла. Порядок в данном множестве определяется сортировкой полей индексных таблиц по значениям поисковых атрибутов и сортировкой каждого из инвертированных списков по значениям ВСН.

В данном случае обеспечивается возможность обращения к первой записи (в смысле порядка, указанного выше), к конкретной записи и к следующей записи множества.

Для обращения к первой записи достаточно указать в условии селекции код имени поискового атрибута. Первой в данном случае считается первая запись первой ассоциации из всех организованных в данном файле ассоциаций по значениям атрибута, указанного в условии селекции. Следует отметить, что в условии селекции в данном случае, как и при обращении к конкретной записи, допускается использовать атрибуты, не входящие в периодическую группу, а также производные атрибуты, не содержащие атрибутов из повторяющихся групп.

Обращение к конкретной записи выполняется по коду имени и значению поискового атрибута. В случае отсутствия заданного значения в записях файла выдается запись с ближайшим большим (в смысле сортировки значений поискового атрибута) значением атрибута. Имеется возможность указания конкретной записи путем задания ВСН, но при этом доступ обеспечивается к записи с ближайшим большим заданного ВСН.

Возможность обращения к следующей записи логически упорядоченного по значениям поискового атрибута множества записей файла позволяет просмотреть все ассоциации данного атрибута с начала или с указанной записи. При этом для каждой очередной прочитанной из БД записи выдается ее ВСН. Следует иметь в виду, что если в условии селекции используется множественный атрибут, то в процессе просмотра одна и та же запись может встретиться несколько раз.

Пример доступа к записям файла, логически упорядоченного по значениям поискового атрибута, представлен на рис. 41. Селекция записей осуществляется с использованием инвертированных списков и позволяет просмотреть все записи ассоциаций записей, организованных по поисковому атрибуту ОК, начиная с заданной записи.

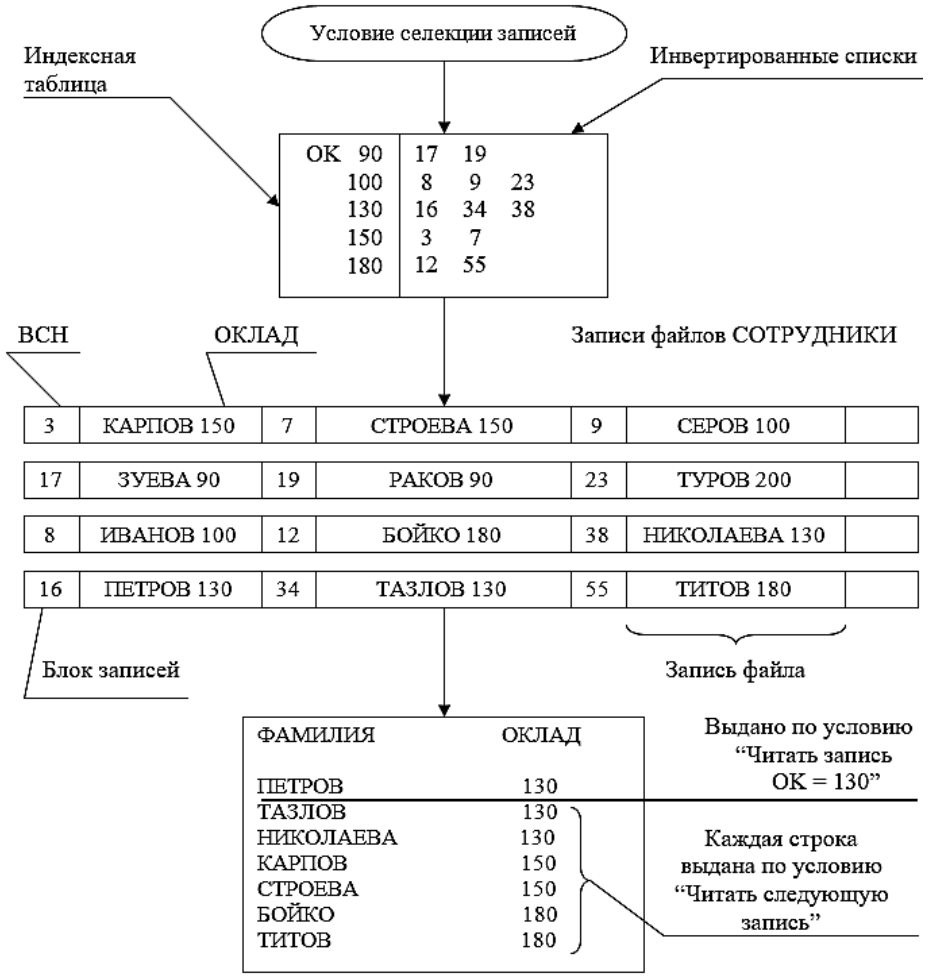


Рис. 41. Доступ к записям файла в логической последовательности по значениям поискового атрибута ОКЛАД (код имени ОК)

Доступ к записям файла в порядке физического размещения записей в БД основан на последовательном считывании блоков физической памяти файла в порядке их расположения в наборе данных. При этом возможно чтение первой записи первого физического блока файла, чтение записи в указанной части памяти файла и чтение записи, физически следующей за указанной.

Поиск записей файла. Поиск записей в Adabas, основанный на использовании ассоциаций, представленных в виде инвертированных списков и списков связи, определяется как ассоциативный поиск.

Ассоциативный поиск обеспечивает возможность селекции записей файла БД по значениям атрибутов, а также селекцию записей файла с учетом их связей с записями другого файла.

Условие селекции определяется в Adabas как условие поиска, которое включает одно или несколько простых условий поиска. Простые условия поиска в пределах условия поиска соединяются логическими операциями И.

Простое условие задает для поискового атрибута (в том числе производного) одно или несколько значений с использованием логических операций И, ИЛИ, ОТ ДО, КРОМЕ. Логическая операция ОТ ДО задает диапазон значений поискового атрибута, исключение значения из которого определяется логической операцией КРОМЕ.

Селекция по значениям атрибутов в Adabas не требует обращения к записям файлов, поскольку каждое простое условие определяет ключи соответствующих ассоциаций записей файла, созданных в процессе загрузки файла. При этом селекция по значениям атрибутов сводится к формированию списка ВСН записей по каждому из простых условий и пересечению этих списков в соответствии с логикой, заданной в условии поиска. Итогом указанных операций является представленное результирующим списком ВСН множество записей файла, удовлетворяющих условию поиска.

Формирование списка ВСН по простому условию заключается в чтении ряда инвертированных списков по коду имени и значениям поискового атрибута, заданным перечислением конкретных значений или границами диапазона значений, и в дальнейшей обработке этих списков путем выполнения необходимых

логических операций пересечения, объединения и пересечения с отрицанием.

На рис. 42 представлена схема ассоциативного поиска с использованием инвертированных списков. Условие поиска состоит из двух простых условий, одно из них задано диапазоном значений атрибутов, а другое — перечнем значений. Результирующий список, полученный как пересечение двух списков, сформированных по простым условиям, содержит четыре ВСН записей, удовлетворяющих условию поиска.

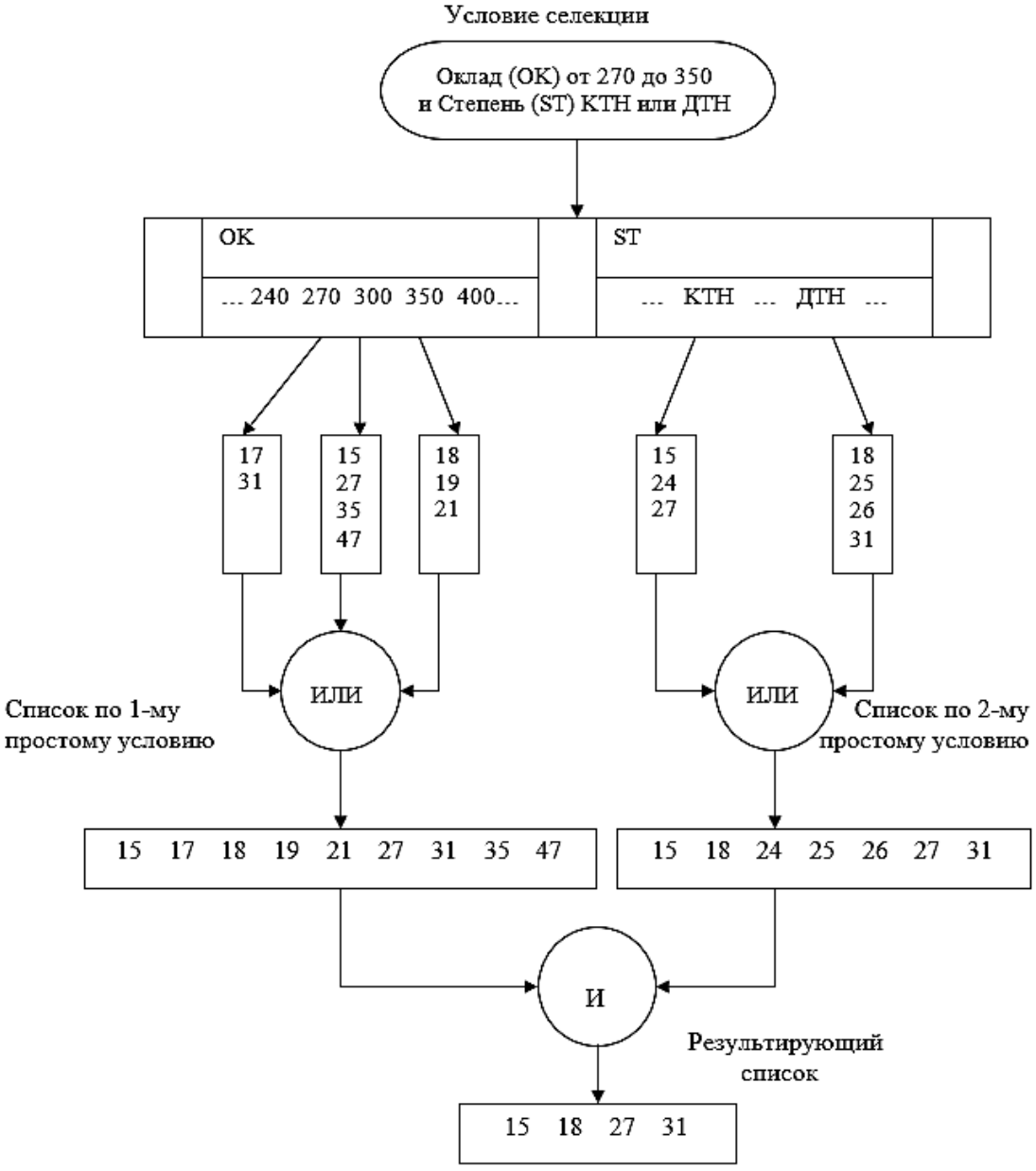


Рис. 42. Ассоциативный поиск записей файла

Если полагать, что чтение каждого инвертированного списка с учетом многоуровневости индексных таблиц требует в среднем четырех операций ввода-вывода, то практически независимо от количества записей файла БД ассоциативный поиск в данном случае потребует 20 операций ввода-вывода, что дает ощутимый выигрыш во времени по сравнению с методами использования цепочек и прямого просмотра файла БД.

При использовании инвертированных списков некоторая зависимость времени поиска от количества записей файла проявляется в больших БД, содержащих 10^5–10^6 записей, поскольку возникает необходимость обработки больших списков ВСН по частям.

Селекция по связям записей файла. Как уже отмечалось, связь между записями двух файлов устанавливается по равенству значений атрибутов связи в этих записях и фиксируется путем построения прямых и обратных списков связи, при этом понятия «исходный» и «связанный» файлы являются относительными.

В модели данных концептуального уровня Adabas реализуются две разновидности селекции по связям между записями файлов:

– по наличию связей записей файла с записью другого файла;

– по значениям атрибутов в записях связанных файлов.

Первая разновидность селекции, называемая поиском связанных записей, использует списки связи для отображения записи одного файла в одну или более записей второго файла. В качестве ключа списка при этом используется ВСН отображаемой записи. Данная разновидность селекции является спецификационной операцией и может быть использована, например, в иерархической структуре для перехода от исходного узла к порожденному без обращения к экземплярам порожденного узла.

Вторая разновидность селекции, называемая поиском в связанных файлах, основана на использовании предыдущей разновидности селекции и селекции по значениям атрибутов. В качестве условия селекции при этом выступает составное условие поиска, включающее до пяти условий поиска, одно из которых относится к исходному файлу, а остальные — к связанным файлам.

Результатом поиска в связанных файлах является список ВСН записей исходного файла, которые удовлетворяют заданному условию поиска и связаны с записями нескольких, но не более четырех файлов с заданными свойствами.

Селекция записей файла по значениям атрибутов связанного файла выполняется в несколько этапов:

– селекция по значениям атрибутов в исходном файле;

– селекция по значениям атрибутов в связанном файле;

– отображение результата поиска в связанном файле на исходный файл;

– получение результирующего списка ВСН путем пересечения списков ВСН исходного файла, полученных в результате операций селекции и отображения.

На рис. 43 представлена схема ассоциативного поиска в связанных файлах. В данном примере используются файлы СТУДЕНТ (файл 1) и СЕМИНАР (файл 2), организация связи между которыми была описана ранее (см. рис. 38−40).

Составное условие поиска в примере на рис. 43 соответствует требованию найти в файле 1 все записи о студентах учебных групп 241 и 242, участвующих в семинарах преподавателей кафедры информационно-вычислительных систем (ИВС). Файл 1 объявлен как исходный, поэтому результат поиска в виде списка из трех ВСН записей относится к файлу СТУДЕНТ.

Возможности ассоциативного поиска в БД системы Adabas могут быть расширены благодаря использованию операций над ассоциациями записей файлов. Эти операции обеспечивают возможность логической обработки и сортировки списков ВСН записей файла, которые могут быть получены в результате ассоциативного поиска или сформированы программой пользователя.

Операции логической обработки используют в качестве операндов два списка ВСН записей, принадлежащих одному файлу. Допускается три вида логических операций — пересечение списков, их объединение и пересечение с отрицанием.

Сортировка списка ВСН может быть выполнена в порядке возрастания значений ВСН в списке, а также в порядке возрастания или убывания значений от одного до трех поисковых атрибутов записей файла, которые представлены своими ВСН в сортируемом списке.

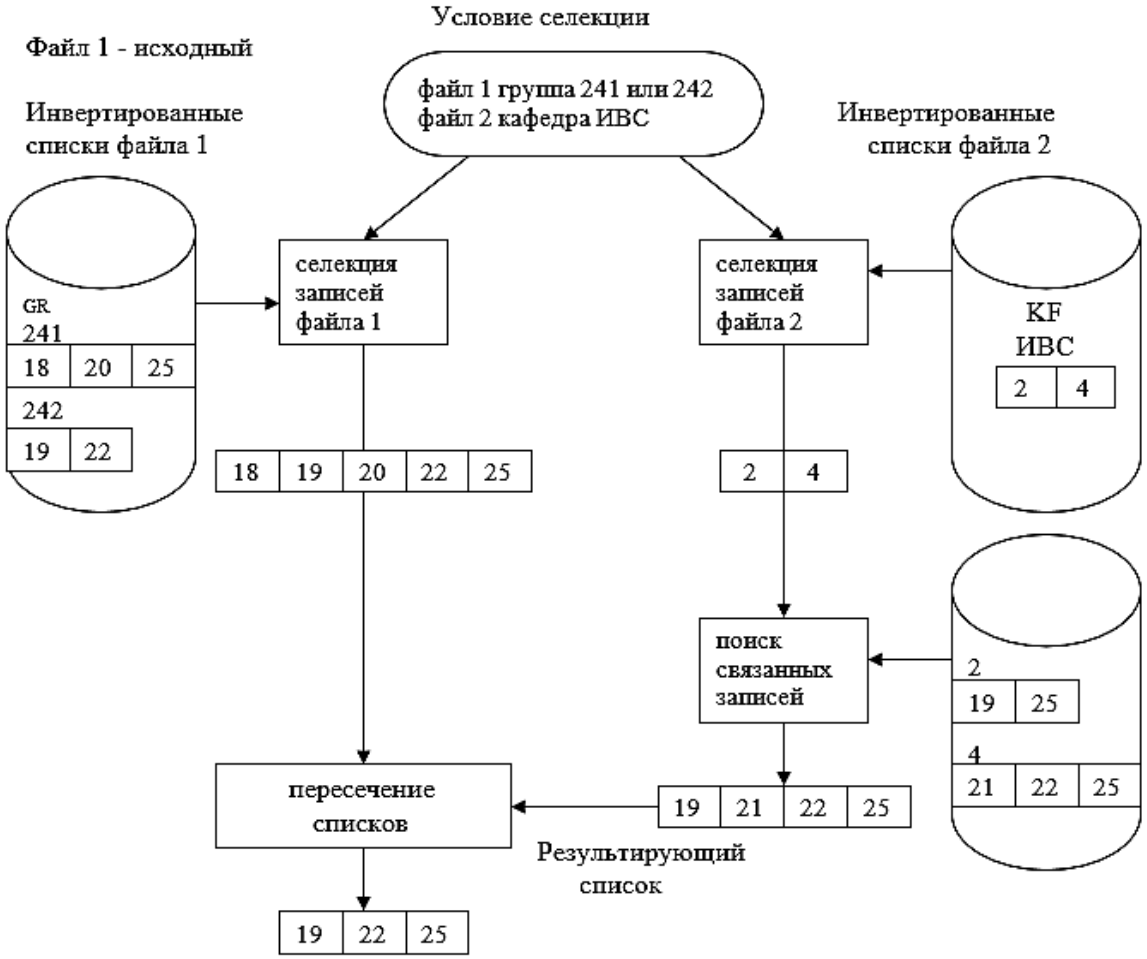


Рис. 43. Ассоциативный поиск записей в связанных файлах

Рандомизированный доступ к записям. Суть метода рандомизации заключается в алгоритмическом преобразовании ключа записи в некоторое число, которое является физическим адресом этой записи. В качестве ключа в Adabas употребляется значение одного из атрибутов записи, которое должно быть уникальным в пределах файла, а физическим адресом записи является относительный номер блока, в котором находится эта запись. Ключом записи может быть также ее BCH.

Алгоритм преобразования ключа в адрес включает нормализацию шестнадцатеричного представления значения атрибута путем его усечения до параметрически заданной длины и деление результата нормализации на простое число, ближайшее меньшее числа блоков экстента набора данных, выделенного для размещения записей данного файла. В качестве физического адреса записи при ее занесении в файл используется остаток от

деления. При рандомизированном доступе физический адрес записи вычисляется алгоритмически по значению ключевого атрибута (или ВСН) точно так же, как это делается при вводе записи. Это означает, что независимо от количества записей в файле рандомизированный доступ к записи требует одной операции ввода-вывода.

При использовании метода рандомизации размещение вводимых записей в пределах файла носит случайный характер и различным ключам записей может соответствовать один и тот же вычисленный физический адрес. Это означает, что в одном блоке набора данных необходимо размещать несколько записей файла. При этом может возникнуть такая ситуация, когда блок, в котором должна быть размещена очередная запись, не имеет достаточно свободного пространства. В подобном случае очередная запись заносится в ближайший свободный блок набора данных и для нее строится обычный инвертированный список по значению уникального атрибута.

Наличие подобных записей в файле ухудшает эффективность рандомизированного доступа к файлу, однако при правильно выбранных параметрах переполнение блоков происходит редко и эффективность этого вида селекции ухудшается незначительно.

Ограничения целостности

Ограничения целостности модели данных концептуального уровня Adabas могут быть разделены на синтаксические и семантические.

Статические синтаксические ограничения, определяющие возможности Adabas по отображению структурных свойств объектов предметной области и отношений между ними в схеме БД, выраженные через максимально допустимые значения параметров ЯОД, приведены ниже:

Число файлов в БД 255
Число записей файла ..... $16{,}7 \cdot 10^6$
Число компонентов схемы файла 500
Число уровней в схеме файла 7
Число экземпляров повторяющейся группы 99
Число экземпляров множественного атрибута 191

Число поисковых атрибутов файла 200
Число элементов составного атрибута 5
Длина значения атрибута:
символьного, байт 253
десятичного целого, разряды 27
упакованного, байт 14
битового, байт 126
двоичного целого, байт 4
символьного или битового поискового, байт 126
производного, байт 256
Число файлов, связанных с данным файлом 18

Динамическим синтаксическим ограничением целостности является запрет на использование в операции модификации более чем одной записи файла БД. Это ограничение минимизирует объем данных, передаваемых между прикладной программой и Adabas в процессе ввода, корректировки или удаления записей файлов БД, что повышает динамичность работы Adabas в режиме обслуживания многих пользователей.

Статические семантические ограничения целостности усиливают действие структурных ограничений, устанавливая дополнительные рамки на значения атрибутов в соответствии с семантикой моделируемых объектов предметной области и тем самым ограничивая число возможных экземпляров объектов, представленных в БД. К данному виду ограничений можно отнести задание на ЯОД числа экземпляров множественного атрибута и периодических групп, а также задание длин значений атрибутов в пределах допустимых максимальных значений, определяемых структурными ограничениями. Возможность определения уникальных атрибутов в схеме файла также относится к числу статических семантических ограничений. Кроме того, статические семантические ограничения в Adabas представлены возможностью контроля по словарю значений символьных атрибутов и контроля по диапазону значений для числовых атрибутов. Требование на указанные виды контроля определяется в словаре данных путем описания атрибутов, подлежащих контролю.

Динамические семантические ограничения, определяющие допустимые переходы БД из одного целостного состояния в другое в зависимости от свойств объектов предметной области, могут быть заданы путем использования специальных команд

ЯМД. К таким командам относятся команды управления транзакциями, с помощью которых определяется допустимая последовательность операций модификации данных, переводящая БД в новое целостное состояние.

5.2. Структура хранения больших данных в СУБД Adabas

Организация среды хранения данных

Среда хранения данных должна обеспечивать эффективное использование вычислительных средств при существующем уровне развития запоминающих устройств. Физическое представление данных и расположение их на запоминающих устройствах определяют физическую организацию данных. Она зависит от используемых методов поиска записей, ввода новых и удаления старых записей.

База данных Adabas размещается на устройствах прямого доступа. Записи БД запоминаются в блоках устройств прямого доступа. Размер блока выбирается с учетом условия эффективного размещения целого числа блоков на дорожке используемых типов устройств прямого доступа.

Размещение записей в блоке обеспечивает более эффективное использование внешней памяти. Для экономии внешней памяти, занимаемой данными, существуют разнообразные методы сжатия:

– исключение из записей полей с отсутствующими значениями атрибутов;

– хранение чисел в упакованном формате;

– подавление повторяющихся символов;

– кодирование часто используемых значений атрибутов;

– посимвольное кодирование и т. д.

Применение того или иного способа представления данных в конкретной СУБД в большой степени зависит от требований, предъявляемых к данной системе, и принятых в ней методов организации поиска данных. Для адекватного отображения структур концептуального уровня на внутренний в системах используются различные вспомогательные данные (указатели, ин-

дексы и т. д.), которые хранятся вместе с данными или отдельно от них. В Adabas принято раздельное хранение вспомогательных данных и самих данных.

Структурными элементами базы данных Adabas на внутреннем уровне являются (рис. 44): накопитель, ассоциатор, рабочий набор и вспомогательные наборы данных.

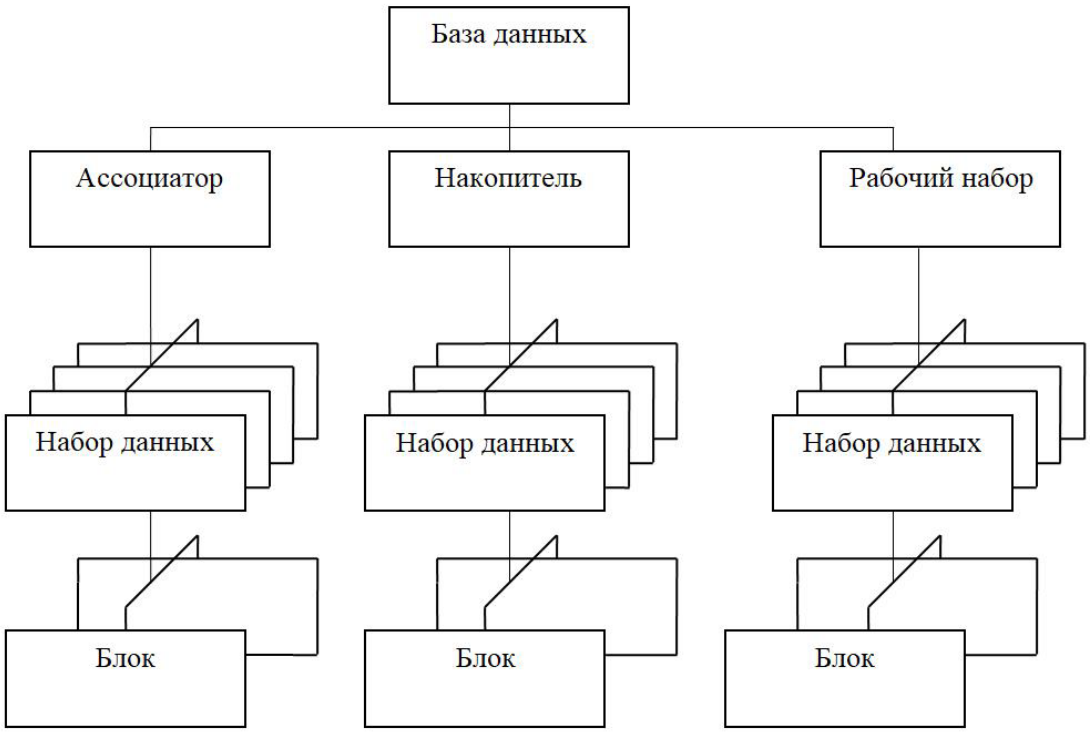


Рис. 44. Структура базы данных

Накопитель, ассоциатор и рабочий набор размещаются в наборах данных, которые имеют прямую организацию и могут быть многотомными. Накопитель и ассоциатор занимают 1–5 наборов данных. Рабочий набор размещается только в одном наборе данных. Набор данных состоит из блоков фиксированной длины. В системе используются также вспомогательные наборы данных: журнал команд, журнал изменений, временный набор, набор для сортировки.

Структура накопителя

Накопитель, который занимает не более пяти наборов данных, предназначен для хранения записей файла БД. Каждому файлу выделяется до пяти экстентов, размещение которых отме-

чается в таблице размещения файла (ТРФ), находящейся в ассоциаторе.

Размер блока зависит от типа устройства внешней памяти. Блок набора данных накопителя состоит из поля длины, содержащего общую длину занятой части блока, и записи файла. Размер записи не должен превышать размера блока. В блоке может находиться несколько записей.

Неиспользованная память в блоках учитывается в таблице свободной памяти (ТСП), находящейся в ассоциаторе, и используется при вводе и корректировке записей.

Запись содержит поле длины и поле ВСН (ISN), за которыми следуют значения атрибутов в соответствии с описанием на ЯОД. ВСН является уникальным ключом записи. Физический адрес блока, в котором находится запись с данными ВСН, определяется посредством преобразователя адреса, размещенного также в ассоциаторе.

Атрибуты представлены в записи с указателями длины значения за исключением атрибутов, определенных на ЯОД с постоянной длиной. Атрибуты без признака «Подавление», значения которых отсутствуют в записи, представляются двумя байтами, первый из которых содержит значение длины, а второй — признак «пустого» значения (пробел для символьных значений и нуль — для числовых).

Атрибуты с признаком «Подавление», значения которых отсутствуют в записи, представлены счетчиком отсутствующих значений. Если в записи содержится подряд несколько таких атрибутов, то счетчик будет показывать их количество. Невведенные значения атрибутов, являющиеся последними в записи, в памяти никак не представляются. Значения множественного атрибута хранятся как последовательность значений с указателем длины и предшествующим однобайтовым счетчиком числа экземпляров. Периодическая группа начинается с однобайтового счетчика, указывающего самый большой номер из хранимых в группе экземпляров.

Изменение значения множественного атрибута на «0» или «_» уменьшает значение счетчика на 1, а изменение значений экземпляра группы на «0» или «_» — не уменьшает.

В процессе работы с БД производятся включение, удаление и модификация записей. Удаление записи вызывает освобождение памяти, которая становится доступной для дальнейшего использования и отмечается в соответствующем поле таблицы свободной памяти. Модификация записи может вызвать уменьшение или увеличение ее размера. Уменьшение длины записи также ведет к освобождению памяти и корректировке поля для соответствующего блока в таблице свободной памяти.

Увеличение длины записи производится за счет свободной памяти в блоке. В случае ее отсутствия запись переносится в блок, имеющий достаточно свободной памяти. Перемещение записей снижает производительность системы, так как требует дополнительных операций ввода-вывода. С целью сокращения количества перемещений записей при их модификации вводится коэффициент заполнения блоков при первоначальной загрузке файла БД. Величина резерва свободной памяти в блоках выбирается исходя из условий использования данного файла. Такой способ размещения записей снижает частоту реорганизации файла.

Вводимая новая запись помещается в блок с достаточным объемом свободной памяти. В случае использования рандомизированного метода доступа запись помещается в блок с требуемым адресом, и если в этом блоке запись разместиться не может, то она помещается так же, как в случае нерандомизированного доступа. Доступ к такой записи будет осуществляться посредством инвертированных списков. Схема размещения записи в блоке показана на рис. 45.

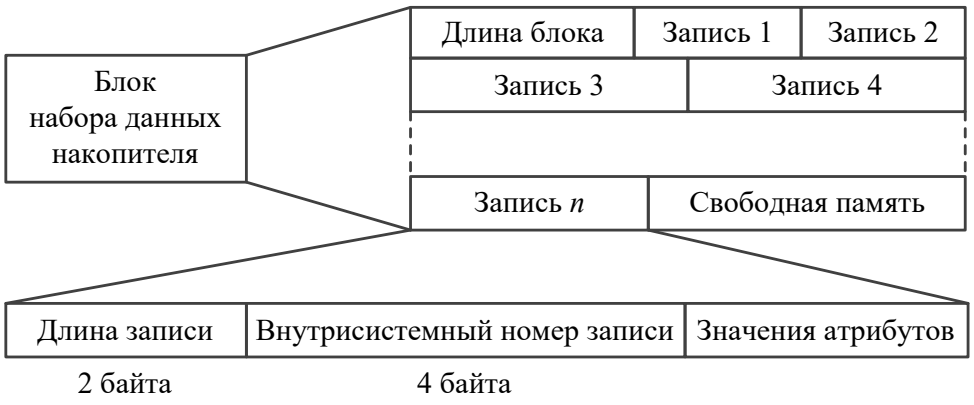


Рис. 45. Размещение записей в блоке

При создании файла БД в накопителе выделяется экстент памяти требуемых размеров для размещения записей.

В процессе эксплуатации система может выделить файлу, по мере необходимости, дополнительно 4 раза по одному экстенту памяти. Увеличение количества экстентов снижает производительность системы. При снижении производительности до неприемлемых значений следует провести реорганизацию базы данных.

На рис. 46 представлена структура записи файла СТУДЕНТ для следующей карточки студента:

НОМЕР КАРТОЧКИ	115573
ФАМИЛИЯ	Ухов
ИМЯ-ОТЧЕСТВО	Петр Иванович
УЧЕБНАЯ ГРУППА	241
СЕССИЯ	1
ЭКЗАМЕНЫ	математика, физика, электротехника
ОЦЕНКИ	5, 4, 4
СЕССИЯ	2
ЭКЗАМЕНЫ	математика, физика, электротехника
ОТМЕТКИ	4, 4, 5

Рис. 46. Структура записи файла СТУДЕНТ

Поля записи (для удобства они пронумерованы) содержат:

1 — общую длину записи;

2 — BCH;

3, 5, 7, 11, 13, 16, 19, 21, 23, 25, 27, 29, 31, 34, 36, 38, 40, 42, 44 — длину значений атрибутов;

4, 6, 8, 9, 12, 14, 17, 20, 22, 24, 26, 28, 30, 32, 35, 37, 39, 41, 43, 45 — значения атрибутов;

10, 18, 33 — счетчики экземпляров множественного атрибута;

15 — счетчик экземпляров повторяющейся группы.

Структура ассоциатора

Ассоциатор содержит сведения о структуре данных концептуального и внутреннего уровней БД в виде таблиц, списков и т. д., помещенных в блоки, и служит для взаимного отображения этих структур и выполнения операций над ними.

Таблицы и списки ассоциатора формируются средствами АБД при создании и ведении БД с помощью словаря данных и модифицируются системой во время ее функционирования. Основная информация о размещении БД сосредоточена в таблице распределения памяти (ТРП).

ТРП занимает один блок ассоциатора и имеет постоянный адрес. Она содержит:

– имя и номер БД;

– максимальное число файлов, которые могут быть загружены в БД;

– число загруженных файлов;

– номера системных файлов;

– адреса экстентов, выделенных для ассоциатора, накопителя, рабочего набора;

– типы устройств, на которых они размещены, неиспользованные области памяти и т. д.

Структура ТРП фиксированная, т. е. каждой характеристике отведено постоянное поле.

Данные о размещении файла в БД содержит таблица размещения файла (ТРФ). Каждому файлу соответствует своя ТРФ, которая занимает один блок ассоциатора. Ее местоположение в ассоциаторе определяется номером файла. ТРФ составляется средствами АБД во время создания файла БД. В ней содержатся:

– имя файла;

– номер файла;

– имя атрибута рандомизированного доступа;

– загрузочная схема связей файлов;

– коэффициент заполнения блоков файла в накопителе;

– коэффициент заполнения блоков инвертированных списков в ассоциаторе;

– сведения о размещении файла в накопителе, инвертированных списков и их индексов в ассоциаторе, преобразователя ВСН записей в адрес блока файла, таблицы учета свободной памяти в блоках накопителя и т. д.

Описание логической структуры файла на ЯОД транслируется в таблицу определения данных (таблица FDT), которая представляет собой загрузочную форму схемы файла.

Таблица FDT для каждого файла имеет фиксированный адрес и занимает 4 блока ассоциатора, поля этой таблицы содержат признаки свойств каждого атрибута файла. Таблица используется программой УВД для взаимного отображения структур концептуального и внутреннего уровней. Прикладным программам обеспечивается доступ к таблице FDT с помощью специальной команды ЯМД.

Блоки накопителя, отведенные под данные, имеют различную степень заполнения. Ввод, корректировка и удаление записей требуют учета свободной памяти в блоках. Учет производится с использованием ТСП. Эта таблица состоит из полей длиной в 1 байт. Номер поля от начала таблицы соответствует номеру блока в накопителе.

Число в соответствующем поле, умноженное на 16, определяет размер свободной памяти (в байтах) в данном блоке накопителя. ТСП формируется при создании БД средствами АБД и модифицируется во время работы системы. Размер ТСП в байтах численно равен количеству блоков накопителя.

Свободные области памяти ассоциатора и накопителя учитываются в таблице свободных областей (ТСО). Она занимает один блок ассоциатора и имеет фиксированный адрес. В этой таблице производится отметка о свободных экстентах памяти накопителя и ассоциатора. Таблица формируется при создании БД средствами АБД и модифицируется во время функционирования системы.

Ряд операций, инициируемых командами ЯМД, выполняется с помощью инвертированных списков, состоящих из ВСН записей файла.

Инвертированные списки хранятся в блоке набора данных ассоциатора совместно с заголовочной частью, состоящей из значения атрибута, которому соответствует список, длины этого

значения и длины инвертированного списка. В одном блоке размещается несколько списков, но все они должны соответствовать значениям одного атрибута. Список отсортирован в порядке возрастания значений ВСН, поэтому первый ВСН оказывается младшим элементом списка.

С целью уменьшения перемещения списков при вводе, корректировке и удалении записей блоки при загрузке файлов заполняются не полностью. Величина резерва задается параметрически при создании файла и может быть изменена утилитой реорганизации ассоциатора в процессе эксплуатации БД. При переполнении блока в результате расширения списка автоматически выделяется новый блок, и часть списка переносится в него так, чтобы в старом блоке остался резерв памяти, установленный АБД.

Доступ к требуемому инвертированному списку осуществляется с помощью многоуровневого индекса, содержащего индекс значений и старшие индексы (рис. 47).

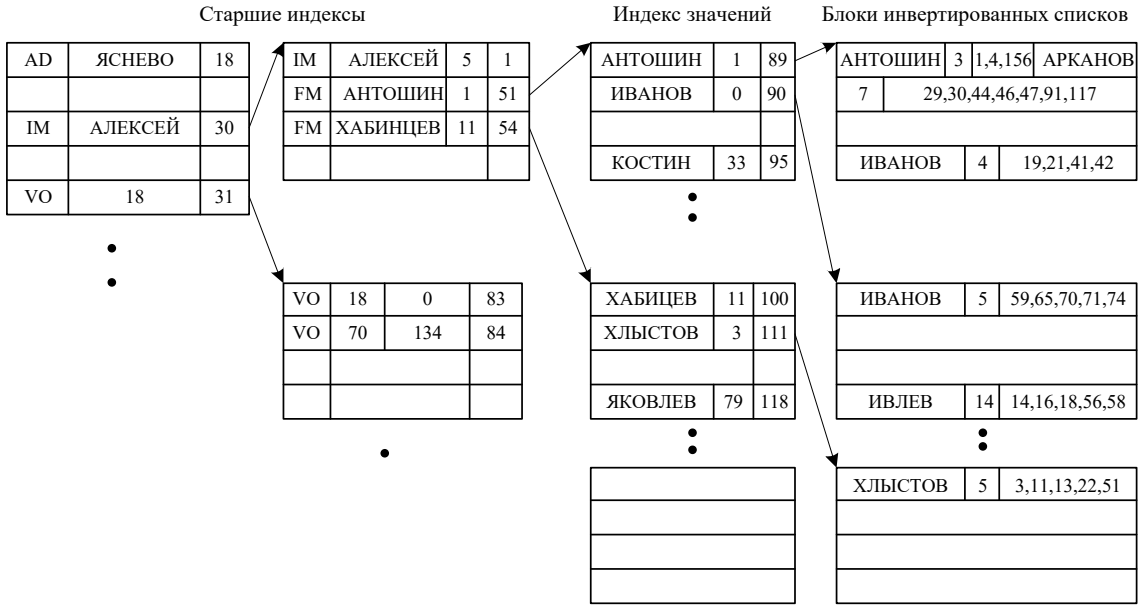


Рис. 47. Инвертированные структуры

Индекс значения состоит из записей, каждая из которых включает длину атрибута, его значение, первый ВСН списка и адрес блока инвертированных списков. Значение атрибута то же, что и в первом инвертированном списке адресованного бло-

ка, а BCH является младшим в этом списке и указывает на то, что список не имеет продолжения. Таких записей в индексе значений будет столько, сколько блоков занимают инвертированные списки данного атрибута. Запись индекса значения, указывающая на блок с инвертированным списком, начало которого находится в предыдущем блоке, вместо младшего BCH списка этого блока содержит 0. Записи индекса значения, относящиеся к определенному атрибуту, объединяются в блоки. В блоке может находиться только целое число записей. Способ выделения новых блоков тот же, что и для инвертированных списков.

Нужный блок индекса значения, относящегося к данному атрибуту, отыскивается посредством старшего индекса. Блок старшего индекса состоит из записей. Каждая запись адресует один блок индекса значений. Запись включает код имени атрибута, поле признаков атрибута, длину значений атрибута, BCH и адрес блока индекса значения. На все поисковые атрибуты заведены записи в старшем индексе. Поисковые атрибуты, не имеющие значений в записях файла, представлены в записях старшего индекса нулевыми значениями полей длины, BCH и адреса блока индекса значений. В поле признаков атрибута указываются характеристики атрибута: формат хранения, принадлежность к повторяющейся группе, признак множественного атрибута.

Каждая запись содержит указатель на один блок индекса значений. Количество блоков старшего индекса зависит от количества блоков индекса значений и в конечном счете — от объема БД. В случае если количество блоков старшего индекса больше одного, то для поиска требуемого блока старшего индекса создается блок старшего индекса следующего уровня иерархии такой же структуры и т. д. Адрес самого верхнего уровня индекса для данного файла указывается в таблице размещения файла.

Связь BCH записи с номером блока в накопителе, в котором находится эта запись, осуществляется посредством преобразователя адреса, который представляет собой таблицу, состоящую из трехбайтовых элементов. Элемент, относительный номер которого равен значению BCH, содержит номер блока накопителя, в котором находится запись с этим BCH. Размер таблицы определяется максимальным значением BCH в данном файле, который задается как параметр при создании этого файла.

При использовании всех выделенных блоков под преобразователь адреса данного файла система автоматически выделяет дополнительный экстент памяти, используя ТСО. Дополнительно система может выделить не более четырех экстентов, после чего требуется реорганизация ассоциатора. Адреса выделенных экстентов содержатся в ТРФ соответствующего файла. Схематически связь ВСН с блоками накопителя посредством преобразователя адреса показана на рис. 48.

Рис. 48. Пример преобразования ВСН в адрес

Структура рабочего и вспомогательных наборов данных

Рабочий набор данных занимает один экстент памяти и состоит из блоков фиксированной длины. Он предназначен для временного размещения промежуточной информации необходимой в процессе работы с БД. В составе рабочего набора выделяется область оперативного журнала изменений, используемого для поддержания логической целостности БД, области промежуточных списков ВСН, а также области результирующих списков ВСН.

Область оперативного журнала используется циклически. Она должна быть достаточно большой, чтобы поместить данные за интервал времени между контрольными точками. Величина этой области задается при запуске системы как параметр. В об-

ласть оперативного журнала помещаются записи, подлежащие корректировке, и откорректированные пользовательские данные, а также таблица значений поисковых атрибутов.

Область промежуточных списков ВСН предназначена для размещения списков ВСН, полученных командами поиска и используемых в логических операциях объединения, пересечения, отрицания. Эти списки управляются с помощью двух таблиц: таблицы результирующих списков ВСН и таблицы адресов блока.

Область результирующих списков ВСН содержит списки ВСН с идентификатором, состоящим из идентификатора команды и идентификатора программы (пользователя). Эти списки могут быть многократно прочитаны при выдаче команды с идентификатором той команды, по которой они были найдены.

Для работы ряда утилит предназначены набор для сортировки и временный набор данных. Эти наборы имеют прямую организацию с фиксированной длиной блока. Набор для сортировки используется при создании и модификации инвертированных списков средствами АБД. Временный набор данных применяется в качестве буферной памяти при загрузке БД для создания инвертированных списков. При функционировании системы эти наборы не используются.

Журнал изменений, в котором фиксируются все изменения, производимые в БД, размещается в последовательном наборе данных. Каждому сеансу работы системы соответствует свой набор данных, идентифицируемый номером сеанса.

Во время работы системы выполняемые команды ЯМД протоколируются в журнале команд. Для журнала команд в системе отведены два прямых набора данных либо один последовательный набор данных. Прямые наборы данных применяются в цикле, т. е. после заполнения второго набора вновь используется первый.

Журнал команд может помещаться и в один последовательный набор данных. Способ размещения журнала и размер прямых наборов данных выбирает АБД при запуске системы исходя из целей использования информации журнала.

Операции на внутреннем уровне

Команды ЯМД выполняются программой УБД с использованием таблиц ТРП, ТРФ, ТОФ, ТСП, ТСО, инвертированных списков и преобразователя адреса. Команды ЯМД вызывают исполнение определенной последовательности элементарных операций. Операции над структурами внутреннего уровня выделены в программе УБД в специальный интерфейсный модуль. Этот модуль обеспечивает также обращение утилит к наборам системы.

Выделение функций, связанных с операциями над структурами внутреннего уровня, в отдельный модуль повышает независимость остальных модулей программы УБД и утилит от операционной системы и физических устройств, содержащих наборы данных. Обращение к интерфейсному модулю осуществляется посредством управляющей таблицы. Ее поля содержат следующие параметры:

– код элементарной операции;

– адреса входных и выходных данных;

– код завершения операции.

Интерфейсный модуль осуществляет открытие и закрытие БД, журнала команд, журнала изменений, временных наборов, производит чтение и запись данных, выдачу сообщений на консоль оператора. Для выполнения этих операций интерфейсному модулю указывается код операции, адрес входных данных или адрес буфера для выходных данных. Интерфейсный модуль обеспечивает обращение к операционной системе для того, чтобы выполнить следующие действия:

– выдать текущую дату и время;

– осуществить запрос на ресурс;

– освободить ресурс;

– произвести запрос на оперативную память;

– освободить оперативную память;

– выдать информацию о томах наборов данных БД;

– выдать информацию о внешних устройствах наборов БД;

– дать специальную информацию для программ автономного ведения данных.

Выполнение любой команды ЯМД сопровождается записью в журнал команд кода команды, времени выполнения, идентификатора команды и т. д. Команды чтения могут требовать

обращений к ассоциатору, накопителю и рабочему набору. Последний нужен в команде чтения только при указании «читать следующий» для получения сохраненного в нем списка BCH. Ассоциатор используется для получения системных таблиц, преобразователя адреса и инвертированных списков, однако чтение блоков, в которых они хранятся, производится только в случае их отсутствия в оперативной памяти. Блок помещается в специально выделенную область оперативной памяти, и в случае полного ее применения производится замещение блока, который давно не использовался. Таким образом, сокращается количество обращений к внешним устройствам памяти. Команды поиска используют рабочий набор для размещения промежуточных списков BCH.

Команды модификации и ввода записи в отличие от команд чтения и поиска дополнительно обращаются к журналу изменений и оперативному журналу: прежде чем модифицированные блоки поместятся в накопитель и ассоциатор, происходит запись этих блоков до изменения и после изменения в журнал изменений и оперативный журнал. Запись в журналы производится немедленно при указании специального признака в командах ввода и модификации или при выдаче команд ET, CL и создании контрольной точки.

5.3. Определение логических и физических структур больших данных в СУБД Adabas

Определение базы данных

В рамках создания информационной системы средствами Adabas и Natural первым необходимым шагом является определение структуры базы данных (хранилища данных). Процедура определения базы данных включает три операции: физическое размещение, создание и редактирование файлов-контейнеров базы данных (файлы с названиями ASSO, DATA, WORK). Для создания новой базы данных следует в главном контекстном окне (DBA Workbench) выбрать меню Database — New (рис. 49).

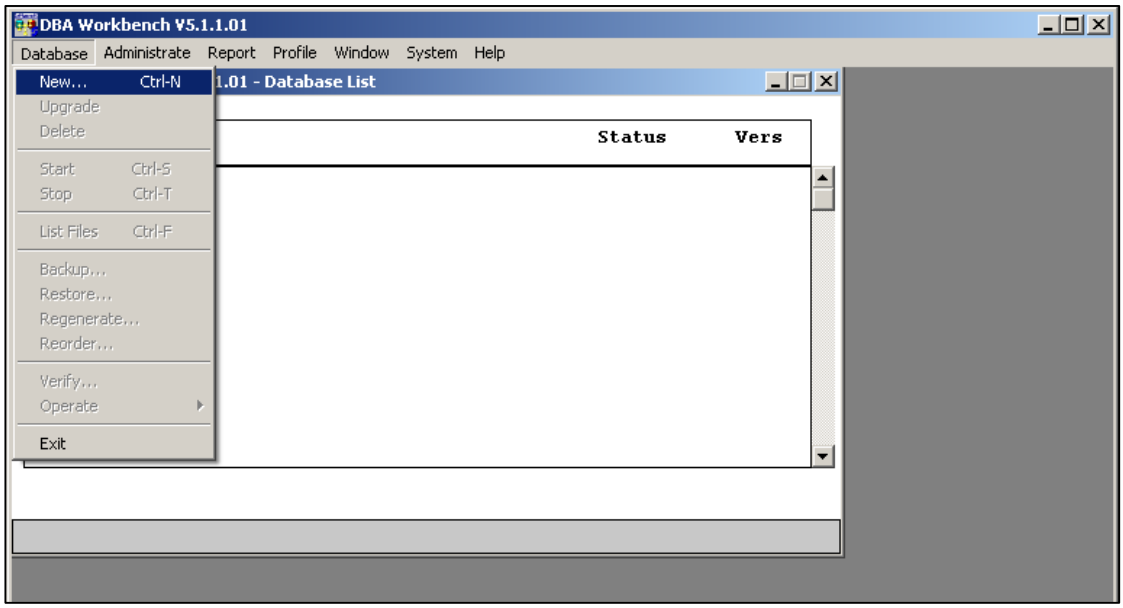


Рис. 49. Создание новой БД

В последующем окне (Create Database) для изменения будет предложен ряд параметров, определяющих размер и свойства файлов-контейнеров и БД (рис. 50). Все значения параметров и БД могут быть сохранены по умолчанию.

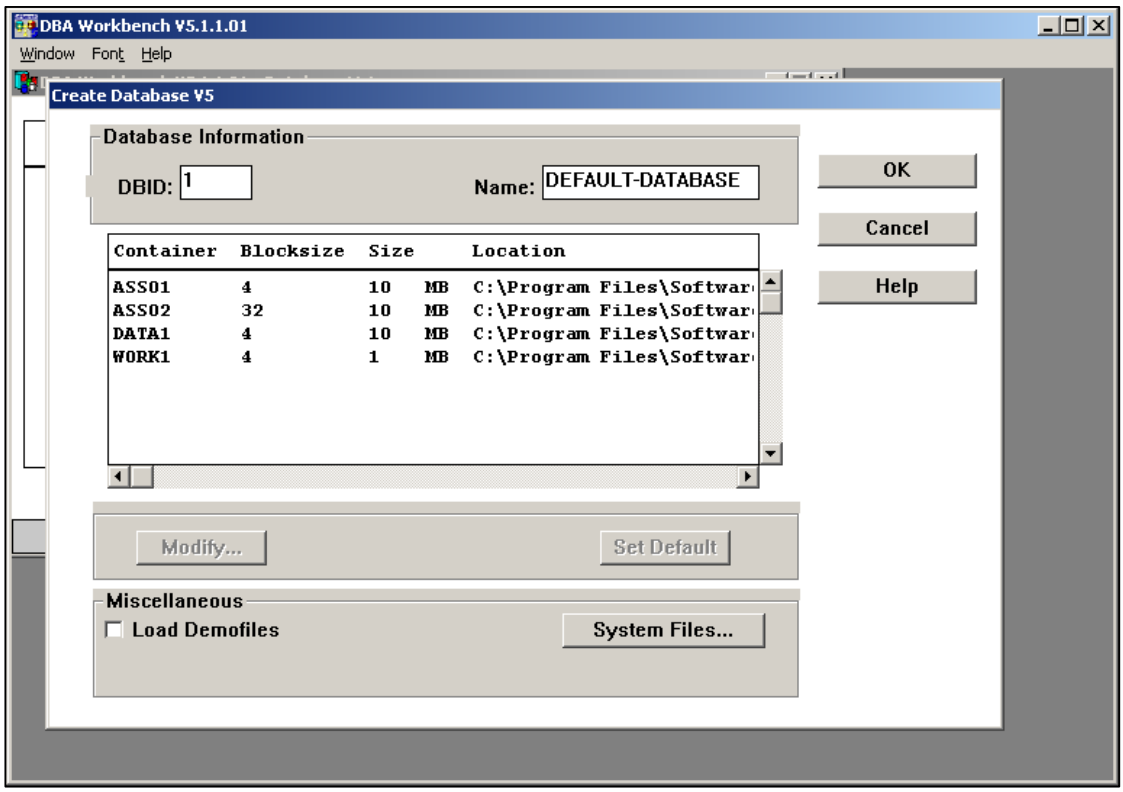


Рис. 50. Параметры БД

Все параметры, предложенные для изменения в появившемся окне, объединены в три группы:

– информация о базе данных (Database Information);

– параметры файлов структуры (ASSO, DATA, WORK);

– прочие параметры (miscellaneous).

Ниже подробно рассмотрены параметры каждой из указанных групп.

Информация о базе данных (Database Information)

Идентификационный номер БД (DBID). При открытии окна (рис. 50) курсор мыши автоматически устанавливается на поле со списком ‘DBID’ (идентификационный номер БД). Следует оставить неизменным номер, предложенный системой, или ввести иной номер при помощи клавиатуры. Также можно воспользоваться списком данного поля и выбрать номер из списка. Следует учитывать, что системой в качестве идентификационного номера БД (‘DBID’) принимается любой номер от 1 до 255, который еще не был использован для создания другой БД.

Название БД (Name). При определении БД в поле ‘Name’ необходимо ввести название БД. Название не должно превышать 16 символов. Также можно сохранить название, предлагаемое по умолчанию.

Параметры файлов структуры (ASSO, DATA, WORK)

Изменение параметров файлов структуры, предложенных системой, следует производить посредством выбора соответствующего файла (ASSO1, ASSO2, DATA1, WORK1 и пр.) при помощи левой кнопки мыши и последующего выбора опции «Редактировать» («Modify…»). При этом системой будет предложено редактирование следующих параметров файлов (рис. 51).

Размер блока файла (Blocksize). В данном поле («Blocksize») вводится размер блока для файла-контейнера ассоциатора БД (единица измерения — килобайт). Если значение данного поля не определено, то системой будет использоваться значение, предложенное по умолчанию. При определении значения данного параметра для файла ассоциатора («ASSO Blocksize») необходимо учитывать, что размер блока файла ассоциатора (ASSO) должен быть меньше размера блока рабочего файла (WORK). В противном случае значение данного поля не будет принято

системой, о чем проектировщик БД будет информирован следующим сообщением (рис. 52).

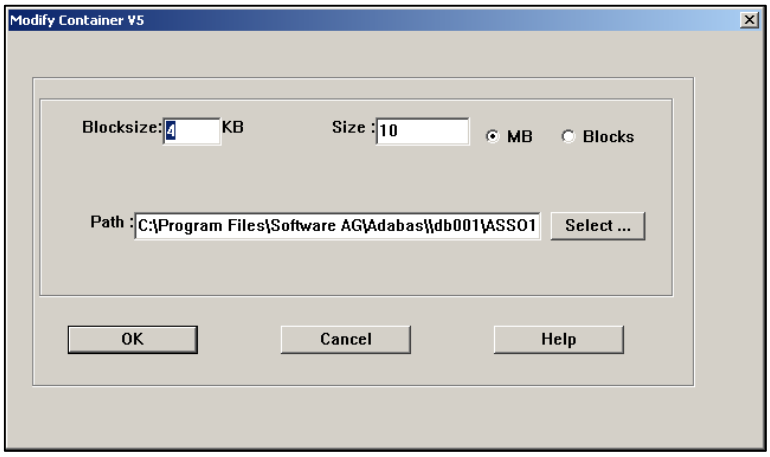


Рис. 51. Изменение параметров файлов структуры

Рис. 52. Информационное сообщение, получаемое при превышении размера файла ассоциатора над размером рабочего файла

Размер файла (Size). Посредством ввода значения в поле «Size» определяется размер дискового пространства, используемого для изменяемого файла. Размер дискового пространства можно определить в мегабайтах (опция «MB») и в блоках (опция «Blocks»). Значение данного параметра, предлагаемое по умолчанию — 1 Мб. Для увеличения дискового пространства для файла ассоциатора необходимо ввести соответствующее значение.

Расположение файла (Path). В текстовом поле «Path» отображается полная спецификация локализации файла ассоциатора (рис. 53). Можно оставить спецификацию неизменной (как предложено системой по умолчанию) либо установить иную область локализации файла при помощи вызова обзорного окна (кнопка «Select», как показано на рис. 51).

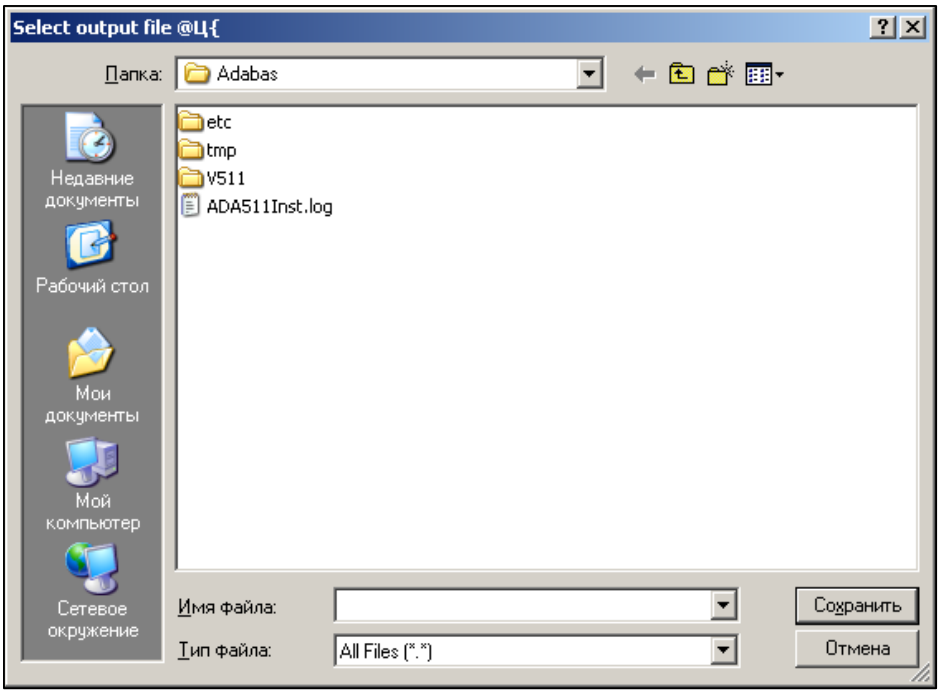


Рис. 53. Определение локализации файла

Для установки параметров файлов структуры на уровне значений, предлагаемых системой по умолчанию, следует воспользоваться опцией «Установить по умолчанию» («Set Default») (см. рис. 50).

Прочие параметры (Miscellaneous)

Номера системных файлов (System Files). В рамках данного подраздела должны быть определены идентификационные номера системных файлов БД (файлы Checkpoint File, Security File, User Data File), которые автоматически создаются и загружаются при создании БД. При нажатии на командную кнопку «Miscellaneous System Files…» появляется следующее окно (рис. 54).

По умолчанию системой предлагаются следующие номера системных файлов: checkpoint — 1, security — 2, user data — 3. При необходимости можно изменить номера системных файлов. Для сохранения изменений нажмите OK. Номера системных файлов не должны превышать максимальное число файлов БД («Miscellaneous Maximum No. of Files»). В противном случае произведенные изменения не будут сохранены системой, о чем проектировщик БД будет проинформирован следующим сообщением (рис. 55).

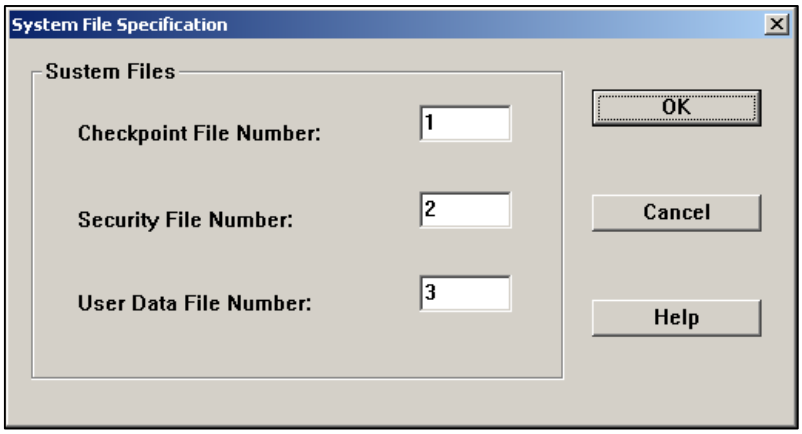


Рис. 54. Прочие параметры

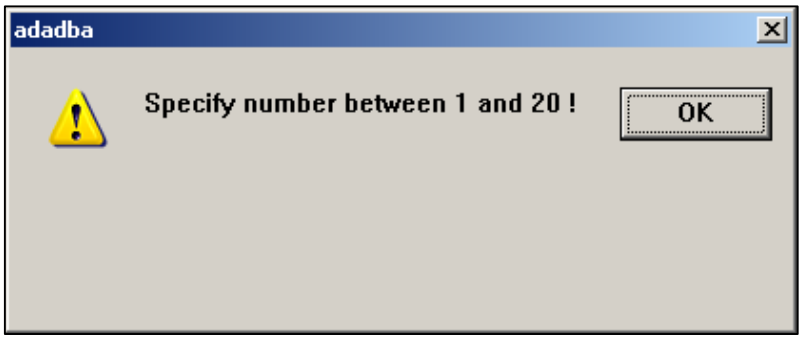


Рис. 55. Информационное сообщение,
получаемое при превышении максимального числа файлов БД

Загрузка демофайлов (Load Demofiles). Для загрузки демонстрационных файлов, предлагаемых системой, следует поставить флажок на опции «Загрузка демофайлов» («Load Demofiles»). При этом после завершения процедуры создания БД в структуру БД будут загружены соответствующие файлы.

После ввода и редактирования указанных параметров (информация о базе данных (Database Information), параметры файлов структуры (ASOO, DATA, WORK), прочие параметры (Miscellaneous)) для определения новой БД следует в окне «Create Database» выбрать ‘OK’.

Если вышеописанные параметры были заданы корректным образом, проектировщик будет проинформирован о создании новой БД следующим сообщением (рис. 56).

Вид главного управляющего окна («DBA Workbench») поменяется следующим образом (рис. 57).

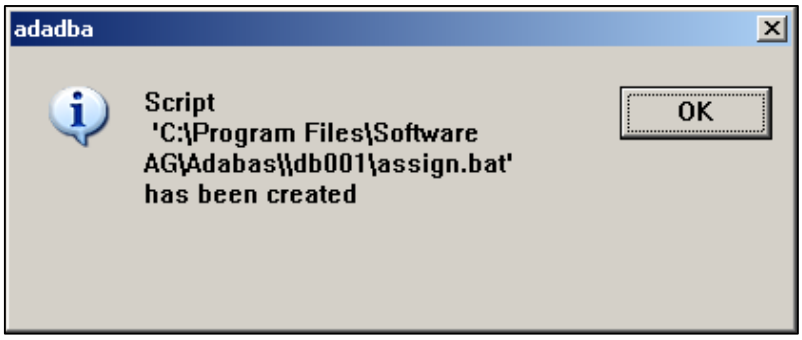


Рис. 56. Информационное сообщение об успешном создании БД

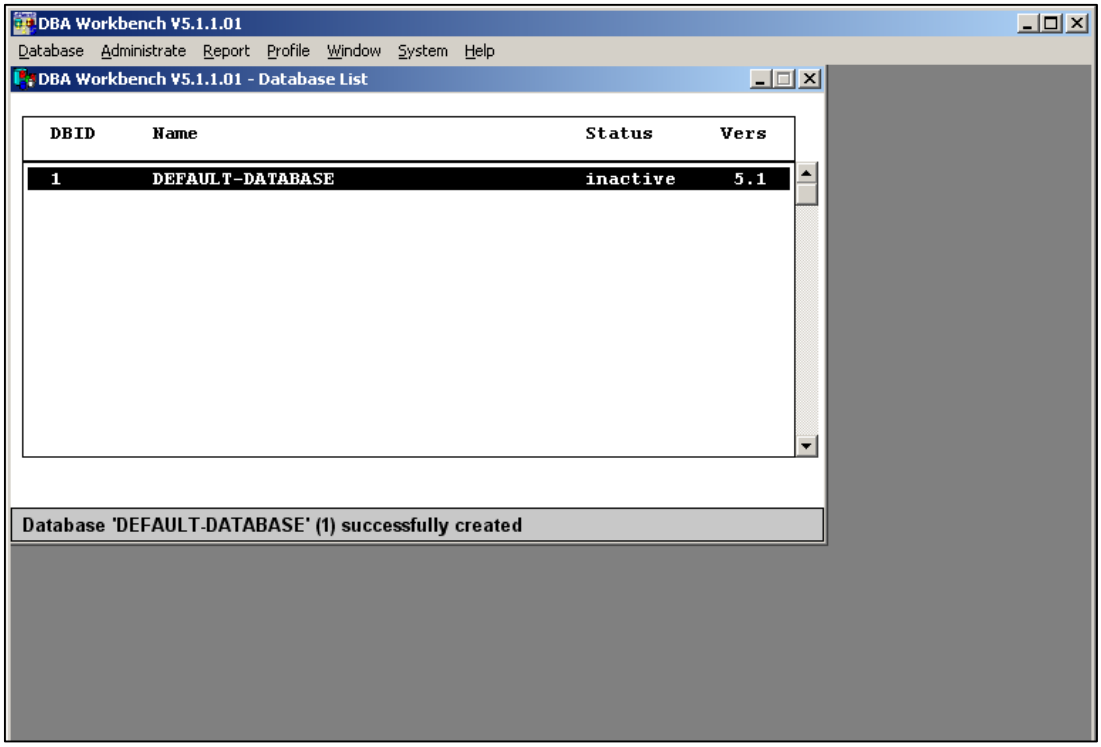


Рис. 57. Вид главного управляющего окна после создания новой БД

5.4. Определение файлов для хранения больших данных в СУБД Adabas

После определения БД для обеспечения возможности осуществления транзакций (операции внесения, удаления, редактирования данных) необходимо определение структуры файла в рамках созданной БД.

В системе Adabas определение файла подразумевает создание таблицы определения данных (Field Definition Table —

FDT) с определением свойств файла (локализация дискового пространства файла и пр.). После создания FDT и определения параметров файл создается в рамках выбранной БД и отображается в списке файлов БД.

Таблица FDT создается при помощи редактора FDT, а также может быть прочитана из уже существующего файла FDT или файла модуля определения данных (Data Definition Module — DDM) конструктора Natural. Для создания нового файла в рамках существующей БД необходимо в главном управляющем окне (DBA Workbench) выбрать (щелчком мыши) БД, в список файлов которой необходимо включить определяемый файл (рис. 58).

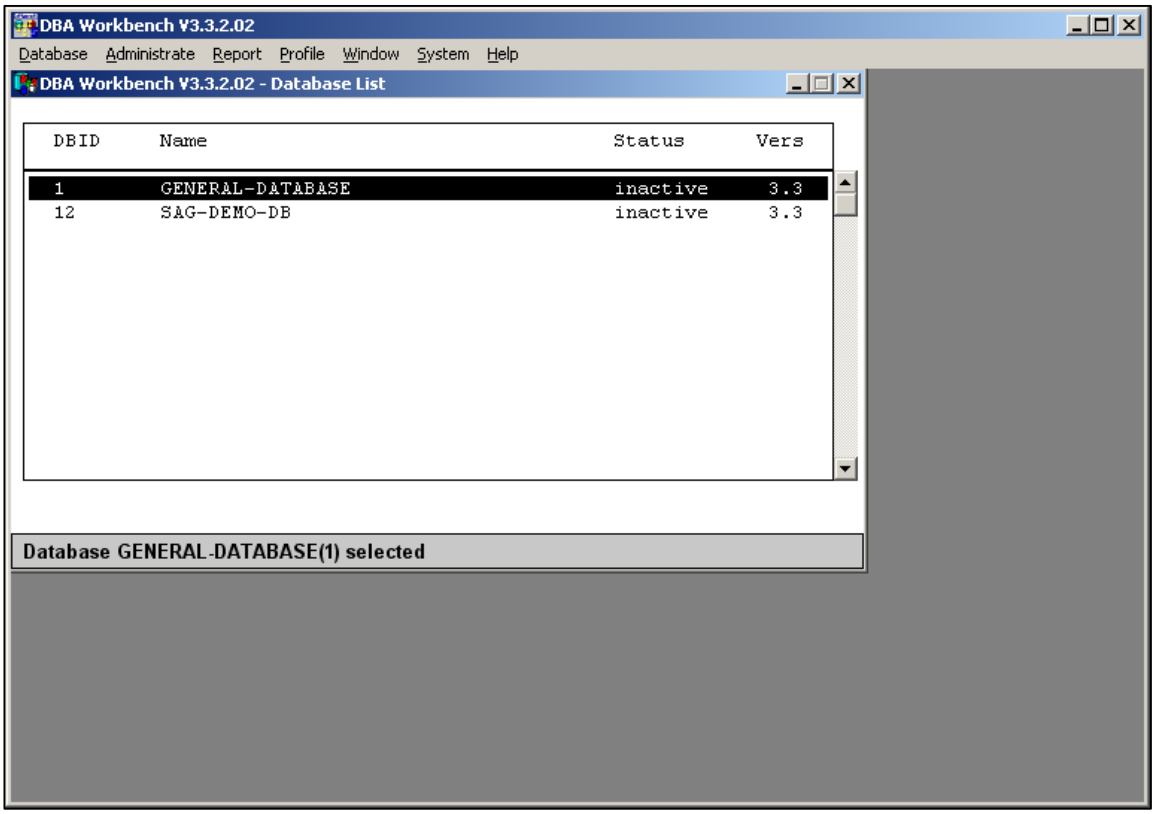


Рис. 58. Выбор БД для создания файла БД

Далее необходимо открыть окно списка существующих файлов данной БД. Окно списка открывается либо двойным щелчком мыши по записи выбранной БД, либо посредством меню Database — List Files (рис. 59).

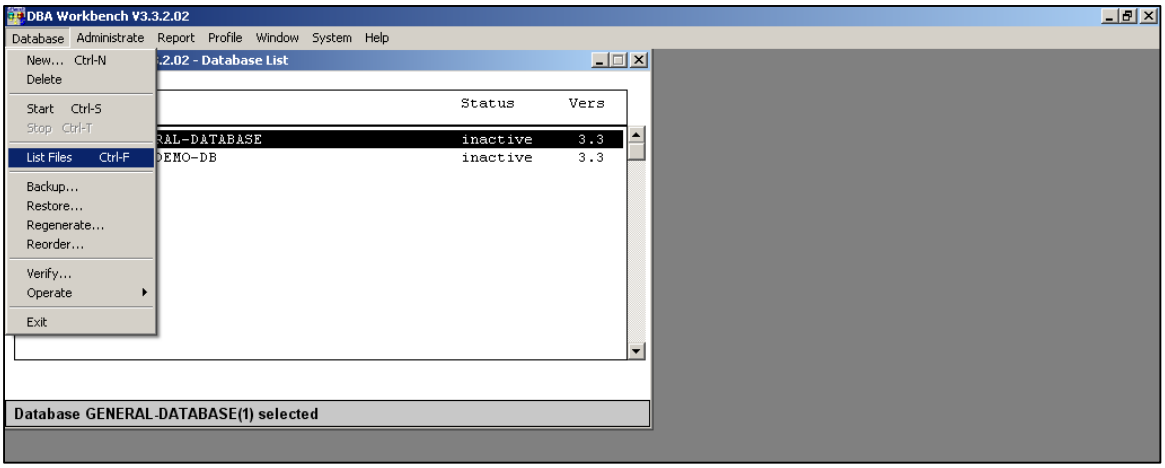


Рис. 59. Просмотр списка уже существующих файлов в БД

В появившемся окне списка файлов, даже если ни одного файла не было создано, будут отображаться 3 системных файла (рис. 60). Данные файлы создаются автоматически при определении БД.

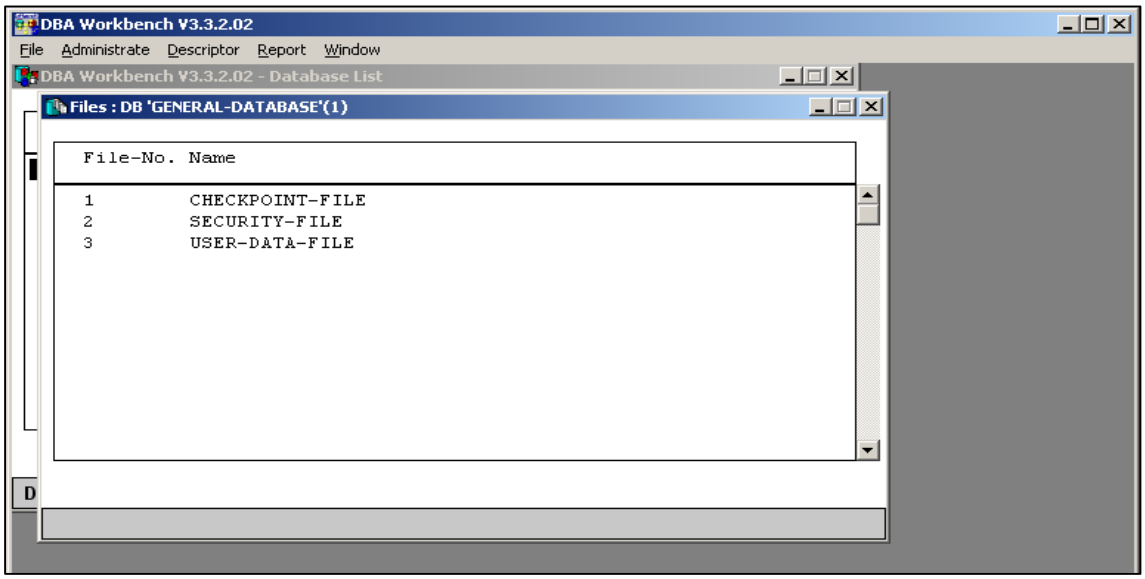


Рис. 60. Список уже существующих файлов БД

Как показано на рис. 60, системные файлы имеют первые три номера. Эти номера присваиваются по умолчанию при определении БД и могут быть изменены на другие посредством «Miscellaneous System Files…».

Далее следует выбрать в контекстном меню File → New (рис. 61).

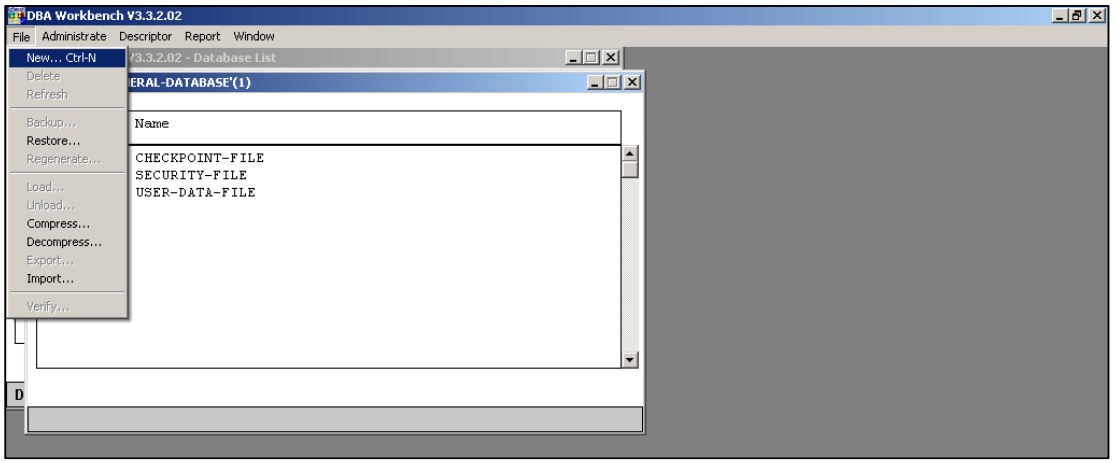


Рис. 61. Создание нового файла БД

После проведения данной процедуры открывается окно редактора FDT (рис. 62). При помощи данного редактора осуществляется определение составляющих файла (записи и элементов) посредством ввода следующей информации в текстовые поля и поля со списком: тип элемента, уровень элемента, сокращенное имя элемента, длина элемента (в байтах), формат данных элемента и опции ограничения элемента записи. Кроме того, любой из элементов, определяемых в рамках файла, может быть объявлен дескриптором (простым дескриптором, фонетическим дескриптором, субдескриптором, супердескриптором или гипердескриптором).

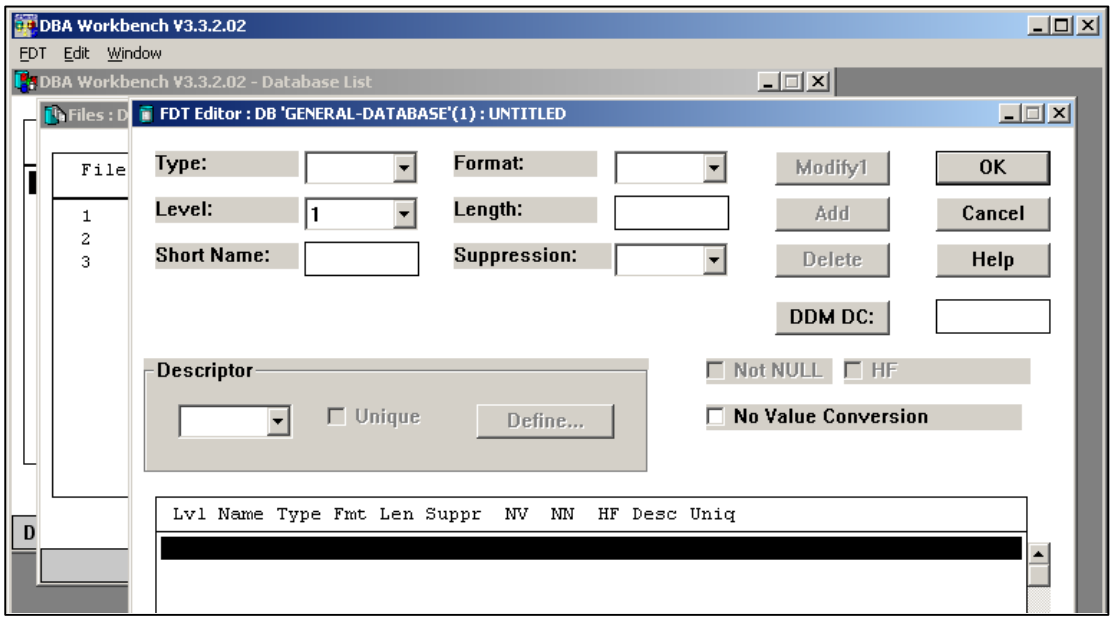


Рис. 62. Окно редактора БД

После определения составляющих файла для сохранения всех введенных значений элементов в структуре файла необходимо нажать ‘OK’. При этом открывается окно ввода параметров необходимых для создания файла (Create File) (рис. 63).

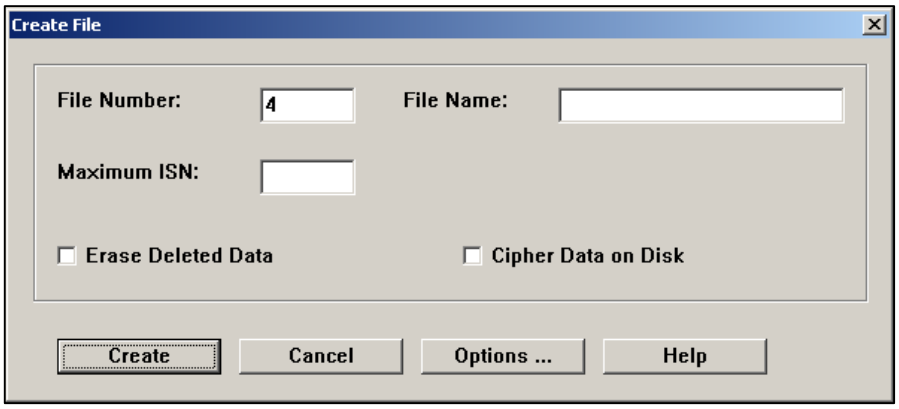


Рис. 63. Сохранение значений элементов в структуре файлов

Для успешного определения файла необходимо ввести ряд следующих параметров в данном окне:

– номер файла (File Number). В данном поле отображается следующий свободный порядковый номер файла. Можно оставить предлагаемое значение либо ввести иной номер. При этом следует учитывать, что номер не должен превышать максимальное число файлов, определенных для данной БД, а также должен быть отличным от номеров уже существующих файлов и номеров системных файлов, отображаемых в списке файлов БД;

– название файла (File Name). Название файла, введенное в данное поле, будет отображаться в списке файлов БД и отчетах;

– максимальный ISN (Maximum ISN). Для дальнейшего определения файла необходимо указать предполагаемый наибольший внутрисистемный номер (Internal Sequence Number — ISN) в текстовом поле ‘Maximum ISN’. Следует учитывать, что от значения данного параметра напрямую зависит размер дискового пространства, которое будет отведено для адресного преобразователя (Address Converter) при создании файла;

– опция «Стереть удаленные данные» (Erase Deleted Data). Данная опция используется для замещения индексов и блоков хранения данных нулевыми значениями при их возвращении к свободному табличному пространству (удалении);

– опция «Шифровать данные на диске» (Cipher Data on Disk). Опция используется для включения функции шифрования данных.

Для создания файла после ввода перечисленных параметров следует выбрать ‘Create’. При этом параметры элементов файла и характеристики структуры данных создаваемого файла будут сохранены установленными по умолчанию. Если все параметры были введены корректно, в списке файлов БД будут отображены номер и имя созданного файла (рис. 64).

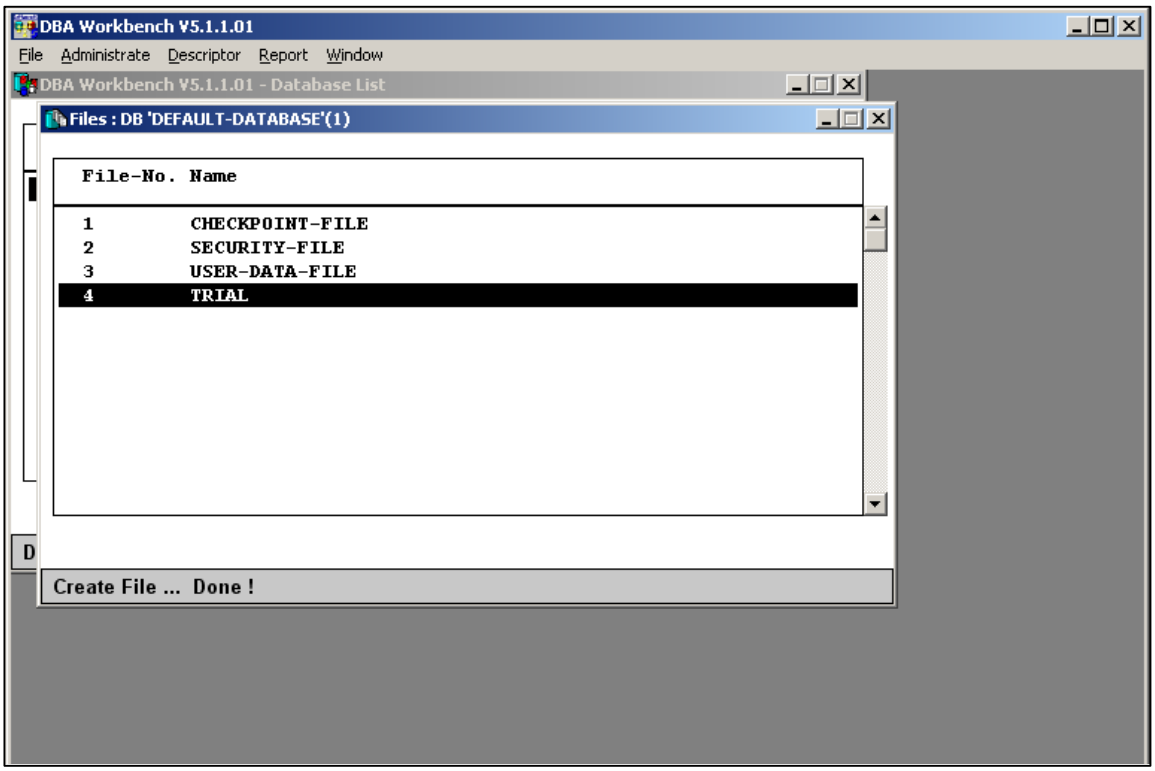


Рис. 64. Список файлов БД после создания нового файла БД

Определение записи в структурах больших данных

Под записью в данном случае понимается совокупность всех полей, определенных в рамках файла. Под определением записи понимается последовательное определение каждого из полей (атрибутов) посредством редактора FDT при определении файла. Каждое из полей (атрибут) определяется рядом обязательных параметров, в то время как часть параметров назначается по выбору. При определении файла в рамках существующей БД проектировщик БД вызывает редактор FDT (рис. 65).

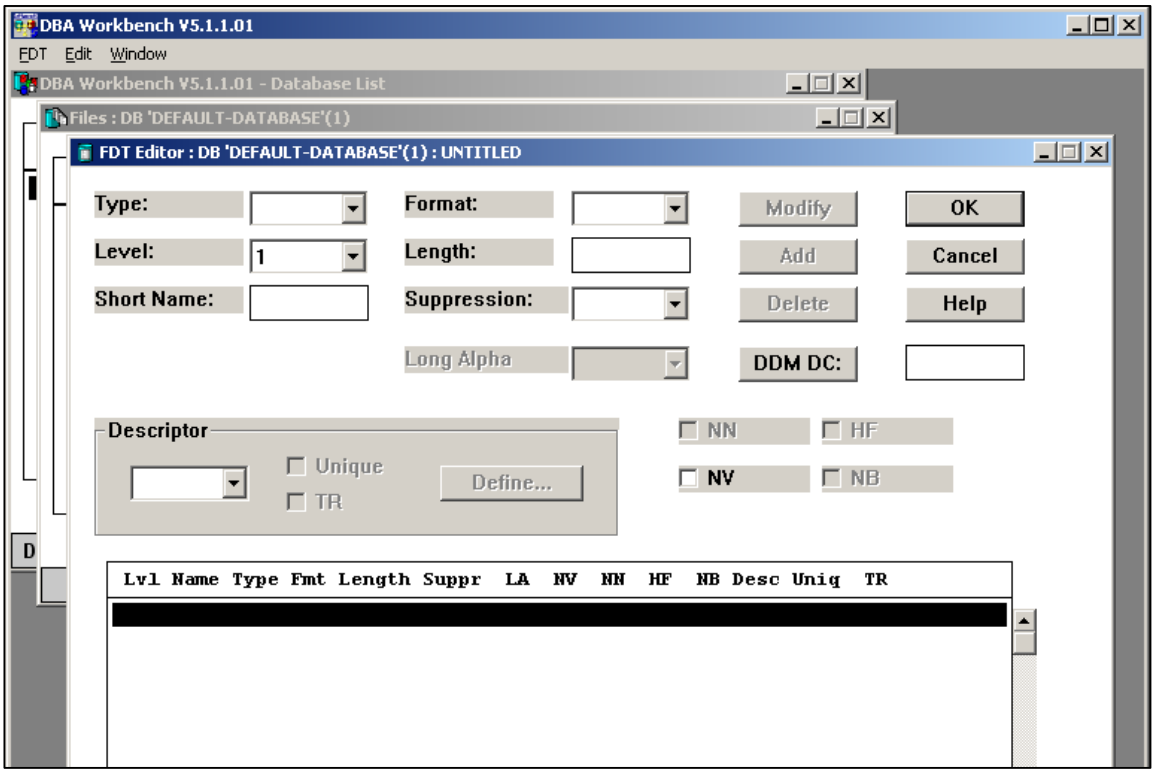


Рис. 65. Создание записи при помощи редактора FDT

Далее для каждого поля (атрибута) при помощи текстовых полей и полей со списком вводятся значения следующих параметров.

Тип (Type). Допускается пустое значение. Следует из списка поля ‘Type’ выбрать необходимый тип определяемого атрибута. По умолчанию системой предлагается пустое значение. Обозначениям данного параметра соответствуют следующие значения (рис. 66):

– пустое значение — простое (элементарное) поле;
– GR — группа;
– MU — множественное поле;
– PE — периодическая группа.

Опция MU указывает, что поле может содержать больше одного значения в отдельной записи. Фактическое количество значений, присутствующих в каждой записи, может меняться от 0 до 191, но по крайней мере одно значение (даже пустое) должно присутствовать в каждой входной записи.

Значения хранятся согласно другим опциям, указанным для поля. Первое значение — это предшествующий полю счет-

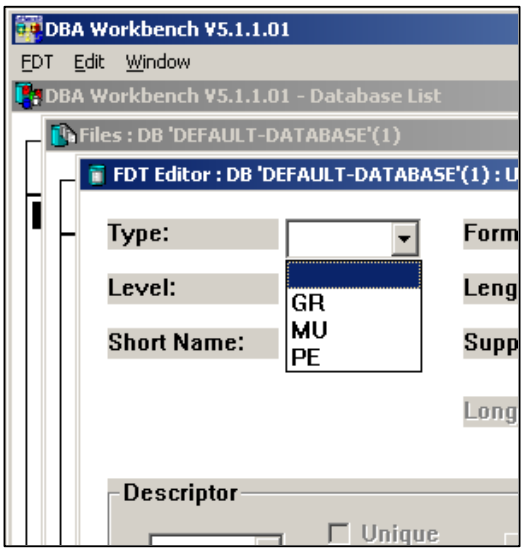


Рис. 66. Тип определяемого атрибута

чик, который указывает количество значений, в настоящее время присутствующих в поле. Количество хранимых значений, равное количеству значений, представленных во входной записи, плюс любые значения, добавленные во время обновления поля, меньше любых подавленных значений (это применяется, только если поле определено с опцией NU).

Если количество значений, содержащихся в каждой входной записи, постоянно, количество может быть определено в описании оператора MU в формате MU(n), где n равняется количеству значений, представленных в каждой входной записи. Например:

FNDEF='01,AA,5,A,MU(3)'

показывает, что три значения множественного поля AA присутствуют в каждой входной записи. Значение ноль (0) указывает, что во входной записи никакие значения для множественного поля не присутствуют.

Если количество значений не постоянно для всех входных записей, то однобайтовый двоичный счетчик поля должен предшествовать первому значению каждой входной записи, указывая количество существующих в ней значений.

Все значения, задаваемые во время входа или обновления, будут сжаты (если только не была указана опция FI). Необходимо проявить осторожность при совместном использовании опций FI и MU: если присутствует большое количество сжимае-

мых значений, то можно потратить впустую большое количество памяти на диске.

Если опция NU определена с опцией MU, то нулевые значения будут подавлены и логически, и физически. Позиционность всех значений (включая пустые значения) поддерживается в MU реализации, если не определена опция NU. Если большое количество нулевых значений присутствует в группе поля MU, опция NU может уменьшать требования дисковой памяти для поля, но не должна использоваться, если должны поддерживаться относительные позиции значений.

Опция NC (или NC/NN) не может быть определена для поля с опцией MU.

Опция PE указывает, что должна быть определена периодическая группа. Периодическая группа может состоять из одного или более полей, и в пределах данной записи могут встречаться от 0 до 99 реализаций (или 191, если определен параметр утилиты ADACMP MAXPE191), но по крайней мере одна реализация (даже если она содержит все нулевые значения) должна присутствовать в каждой входной записи.

Периодическая группа должна быть описана на уровне 01. Все поля, которые должны содержаться в периодической группе, должны следовать тотчас после и должны быть описаны на уровне 02 или выше (с приращением 1 до максимального 7 уровня). Следующее описание уровня 01 указывает на конец текущей периодической группы.

Опция PE может быть описана только с именем группы. Параметры длины и формата не могут быть определены с именем группы. Периодическая группа может содержать дескрипторы и/или множественные поля и другие группы, но не может содержать другую периодическую группу. Далее следуют примеры:

```
FNDEF='01,GA,PE'              периодическая группа GA
FNDEF='02,A1,6,A,NU'
FNDEF='02,A2,2,B,NU'
FNDEF='02,A3,4,P,NU'
FNDEF='01,GB,PE(3)'           периодическая группа GB
FNOEF='02,B1,4,A,DE,NU'
FNDEF='02,B2,5,A,MU(2),NU'
```

FNDEF='02,B3'
FNDEF='03,B4,20,A,NU'
FNDEF='03,B5,7,U,NU'

Номер уровня (Level)

Не допускается пустое значение. Значение, устанавливаемое по умолчанию — 1. Область допустимых значений — 1−7. Ввести необходимое значение уровня можно либо с помощью клавиатуры, либо выбрав соответствующий уровень из списка. Номера уровней нельзя пропускать, когда назначаются номера уровней для вложенных групп.

Номер уровня — это число, состоящее из одной или двух цифр в диапазоне 01−07 (лидирующие нули необязательны), используемое для построения группы полей. Поля, имеющие номер уровня 02 или больше, входят в состав предшествующей группы, которой назначен более низкий номер уровня.

Описание группы позволяет ссылку на ряд полей (может быть только 1 поле), используя имя группы. Это обеспечивает удобный и эффективный метод ссылки на ряд логических полей.

Для определения группы могут использоваться номера уровней 01−06. Группа может состоять из других групп. Номера уровней нельзя пропускать, когда назначаются номера уровней для вложенных групп.

```
FNDEF='01,GA'     группа
FNDEF='02,A1,...' элементарное или множественное поле
FNDEF='02,A2,...' элементарное или множественное поле
FNDEF='01,GB'     группа
FNDEF='02,B1,...' элементарное или множественное поле
FNDEF='02,GC'     группа (вложенная)
FNDEF='03,C1,...' элементарное или множественное поле
FNDEF='03,C2,...' элементарное или множественное поле
```

Поля A1 и A2 находятся в группе GA. Поле B1 и группа GC (состоящая из полей C1 и C2) находятся в группе GB.

Сокращенное имя (Short Name)

Не допускается пустое значение. По умолчанию значение не устанавливается. Имя должно быть уникальным в пределах файла, а также должно быть двухсимвольным. Первый символ

обязательно алфавитный, второй символ может быть как символьным, так и числовым. Специальные символы не допускаются. Также нельзя использовать значения E0–E9, так как они зарезервированы системой в качестве редактирующих масок.

Правильные имена: AA, B4, S3, WM

Неверные имена:

A (один символ)

E3 (маска редактирования)

F* (присутствует специальный символ)

6M (первый символ цифра)

Формат (Format)

Не допускается пустое значение. По умолчанию значение не устанавливается. В поле со списком (рис. 67) предлагаются следующие значения параметра «Формат»:

A — алфавитно-цифровой (выровненный влево);

B — двоичный (выровненный вправо, без знака/положительный);

F — с фиксированной точкой (выровненный вправо, со знаком, с использованием двойного дополнения для отрицательных чисел);

G — с плавающей точкой (нормализованная форма, со знаком);

P — упакованный десятичный (выровненный вправо, со знаком);

U — распакованный десятичный (выровненный вправо, со знаком).

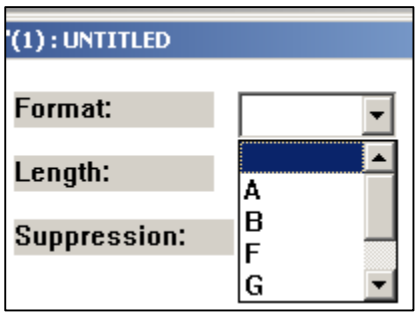


Рис. 67. Формат определяемого элемента

Стандартный формат (по умолчанию) используется Adabas во время обработки команды. Он вводится в таблицу описания

полей и используется при чтении/обновлении поля до тех пор, пока пользователь не определит значение, перекрывающее стандартный формат.

Стандартный формат определяется для поля. Он не может быть определен для имени группы. Когда группа читается (записывается), поля в пределах группы всегда возвращаются (должно быть обеспечено) согласно стандартному формату каждого индивидуального поля. Указанный формат определяет тип сжатия, которое будет выполнено для данного поля.

Поле с фиксированной точкой имеет длину два или четыре байта. Положительное значение находится в нормальной форме, а отрицательное значение в форме двойного дополнения.

Формат поля с плавающей точкой может быть длиной четыре байта (одноразрядное) или восемь байт (двухразрядное). Поддерживается преобразование значения поля, определенного с плавающей точкой, в другой формат.

Если двоичное поле должно быть дескриптором и поле может содержать положительные и отрицательные числа, то вместо формата B должен использоваться формат F, потому что формат B принимает эти значения без знака (как положительные).

Длина (Length)

Допускается пустое значение для имени группы. При определении простого поля указание длины обязательно. По умолчанию значение не устанавливается.

Значение длины используется для того, чтобы определить стандартную длину (по умолчанию), которую использует Adabas во время обработки команды. Указанная стандартная длина вводится в таблицу описания полей (FDT) и используется при чтении/обновлении поля до тех пор, пока пользователь не определит значение, перекрывающее ее.

Максимальные длины полей, которые могут быть определены, зависят от формата (табл. 21).

Для имени группы не может быть определена стандартная длина. Стандартная длина не ограничивает размер любого данного значения поля, если только не используется опция FI. Команды чтения или добавления могут перекрывать стандарт-

Таблица 21

Максимальная длина полей в зависимости от формата

Формат	Максимальная длина
Алфавитно-цифровой (A)	253 байта
Двоичный (B)	126 байтов
С фиксированной точкой (F)	4 байта (всегда 2 или 4 байта)
С плавающей точкой (G)	8 байтов (всегда 4 или 8 байтов)
Упакованный десятичный (P)	15 байтов
Распакованный десятичный (U)	29 байтов

ную длину поля до максимальной длины, разрешенной для этого формата. Если стандартная длина поля нулевая, то поле принимается как поле переменной длины. Поля переменной длины не имеют стандартной (по умолчанию) длины. В этом случае длина поля будет принимать значения: для полей с фиксированной точкой (F) — только два или четыре байта; для полей с плавающей точкой (G) — только четыре или восемь байтов.

Если во время выполнения команды Adabas производится обращение к полю переменной длины без указания длины, то будет возвращено значение этого поля, которому предшествует однобайтовая двоичная длина поля (включая байт длины непосредственно). Это значение длины должно быть определено при обновлении поля и также во входных записях. Если поле определено с опцией длинное алфавитно-цифровое (long alpha — LA), значению предшествует двухбайтная двоичная длина поля (включая два байта длины).

Подавление (Suppression)

Допускается и устанавливается по умолчанию пустое значение. Параметры данного поля определяют системные функции подавления нулевых и прочих значений при вводе записи в созданную БД. В поле со списком (рис. 68) предлагаются следующие значения параметра:

– пустое значение — подавление по умолчанию;
– FI — фиксированная длина пространства памяти;
– NU — подавление нулевого значения;
– NC — SQL опция нулевого значения.

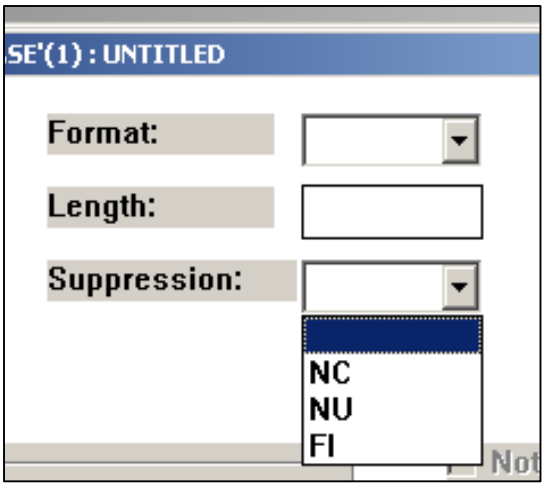


Рис. 68. Функция подавления

FI — фиксированная длина пространства памяти. Опция указывает, что поле должно иметь фиксированную длину пространства памяти. Значения в поле хранятся без внутреннего байта длины, не сжаты и не могут быть длиннее, чем заданная длина поля.

Опция FI рекомендуется для полей длиной в один или два байта, которые имеют низкую вероятность содержания пустого значения (количество персонала, пол и т. д.), и для полей, содержащих значения, которые не могут быть сжаты. Опция не рекомендуется для множественных полей или для полей в пределах периодической группы. Любые пустые значения для таких полей не подавляются (или сжимаются), что может вызвать значительные затраты дисковой памяти и увеличение времени обработки.

Опция FI не может быть определена для:

– полей U-формата;

– полей с опций NC, NN или NU;

– полей переменной длины, определенных с нулевой длиной (0);

– дескрипторов внутри периодической (PE) группы.

Поле, определенное с опцией FI, не может быть обновлено значением, которое превышает стандартную длину поля. Пример использования опции FI показан в табл. 22.

Таблица 22

Пример использования опции FI

Вариант использования	Описание	Данные пользователя	Внутреннее представление
Без опции FI	FNDEF='01,AA,3,P'	33104C 00003C	0433104F (4 байта) 023F (2 байта)
С опцией FI	FNDEF='01,AA,3,P,FI’	33104C 00003C	33104F (3 байта) 00003F (3 байта)

NU — подавление нулевого значения. Опция подавляет нулевые значения, встречающиеся в поле.

Нормальное сжатие (опции NU или FI не определены) приводит к представлению нулевого значения двумя байтами (один байт для значения длины, второй — непосредственно для значения, в этом случае нулевого). Применение опции подавления нулевой величины приводит к внутреннему представлению нулевой величины индикатором однобайтного «нулевого поля». Само нулевое значение не хранится.

Ряд последовательных полей, содержащих нулевые значения и определенных с опцией NU, представляются однобайтовым «пустым полем» индикатором (двоичное представление 11nnnnnn), где nnnnnn — количество соседних байтов внутри поля, содержащих нулевые значения, общим числом до 63. По этой причине поля, определенные с опцией NU, должны быть по возможности сгруппированы вместе.

Если опция NU определена для дескриптора, то все нулевые значения для этого дескриптора не хранятся в инвертированном списке. Поэтому результатом команды FIND (редактор Natural), в которой используется этот дескриптор и для которой нулевое значение используется для поиска, всегда будет отсутствие выбранных записей. Даже в том случае, если в памяти данных имеются записи, которые содержат нулевое значение для дескриптора. Если дескриптор, определенный с опцией NU, используется для управления логической последовательностью в команде чтения в логической последовательности (L3/L6), записи, которые содержат нулевое значение для дескриптора, не будут читаться.

Дескриптор, который нужно использовать в качестве основы для связи файла и для которого существует большое количество пустых значений, должен быть определен с опцией NU, чтобы уменьшить общий размер списков связи.

Опция NU не может быть определена для полей, определенных с объединенными опциями NC/NN или с опцией FI.

Пример использования опции NU (табл. 23) — C1 указывает на 1 пустое поле.

Таблица 23

Пример использования опции NU

Вариант использования	Описание	Данные пользователя	Внутреннее представление
Нормальное сжатие	FNDEF='01,AA,2,B'	0000	0200 (2 байта)
С опцией FI	FNDEF='01,AA,2,B,FF	0000	0000 (2 байта)
С опцией NU	FNDEF='01,AA,2,B,NU'	0000	C1 (1 байт)

NC — SQL опция нулевого значения. В поле, которое не определено с опцией NC (не подсчитывается), нулевое значение является или нулем, или пробелом, в зависимости от формата поля.

В поле, которое определено с опцией NC, нули или пробелы, определенные в буфере записи, интерпретируются по-разному в зависимости от значения «нулевого индикатора»: или как истинные нули или пробелы (как «значащие» нули), или как неопределенные значения (как истинный SQL или «незначащие» нули).

Если поле, определенное с опцией NC, не имеет никакого значения, указанного в буфере записи, значение поля всегда обрабатывается как SQL нулевое значение.

Когда интерпретируется как истинный SQL нуль, нулевое значение удовлетворяет SQL интерпретацию поля, не имеющего никакого значения. Это означает, что значение поля не было введено; т. е. значение поля не определено.

Значение нулевого индикатора ответственно за внутреннее Adabas представление нуля. Это значение всегда имеет длину два байта и формат с фиксированной точкой, независимо от формата данных. Он определяется в буфере записи, когда значе-

ние поля добавляется или обновляется, и возвращается в буфер записи, когда значение поля читается.

Для команд Update (обновления) (Ax) или Add (добавления) (Nx) значение нулевого индикатора должно быть установлено в буфере записи в положение, которое соответствует обозначению полей в форматном буфере. Установка (шестнадцатеричное значение) должна быть одной из следующих:

FFFF — значение поля установлено в «не определено», незначащий нуль; различия между «нет значения», двоичные нули или пробелы для поля в буфере записи игнорируются; все интерпретируются одинаково как «нет значения»;

0000 — нет значения, двоичные нули или пробелы для поля в буфере записи интерпретируются как незначащие нулевые значения.

Для команд Read (чтения) (Lx) или Find с Read (поиска с чтением) (Sx с форматным буфером) ваша программа должна проверять значение нулевого индикатора, возвращенного в позицию буфера записи, соответствующую позиции поля в форматном буфере. Нулевой индикатор может принимать следующие значения:

FFFF — ноль или пробел в поле не важны;

0000 — ноль или пробел в поле — значащее значение, т. е. истинный ноль или пробел;

xxxx — поле усечено, значение нулевого индикатора содержит длину (xxxx) полного значения, которое хранится в записи базы данных.

«Длинные значения» (Long Alpha)

Предлагаются следующие опции для данного параметра (рис. 69):

L4 — длинное алфавитно-цифровое поле (4 байта);

LA — длинное алфавитно-цифровое поле (2 байта).

Опции L4 и LA могут использоваться для полей алфавитно-цифрового формата. При этом такое поле может содержать значение длиной до 16,381 байт.

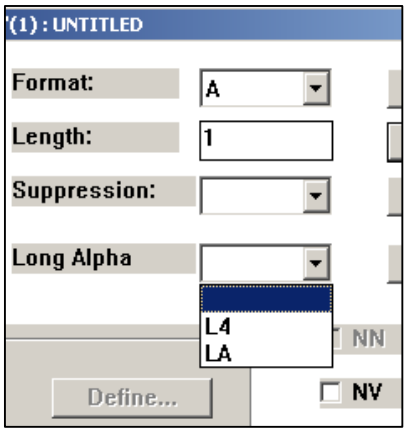


Рис. 69. Функция Long Alpha

Дескриптор (Descriptor)

Допускается и устанавливается по умолчанию пустое значение. Любое поле при определении FDT может быть определено дескриптором. Для поля, определенного дескриптором, будет сделан вход в инвертированном списке ассоциатора, который позволит этому полю: использоваться в выражении поиска и в качестве ключа сортировки в команде FIND, управлять последовательным логическим чтением или использоваться для связи файла.

Если дескрипторное поле не инициализировано и логически падает мимо конца физической записи, вход инвертированного списка для той записи не формируется по причине производительности. Опция дескриптора должна использоваться осторожно, особенно если файл большой и поле, которое рассматривается как дескриптор, часто обновляется.

В процессе определения записи предлагаются следующие опции для параметра «Дескриптор» (рис. 70):

DE — дескриптор;

SB — субдескриптор;

SP — супердескриптор;

HY — гипердескриптор;

PH — фонетический дескриптор.

При определении всех типов дескриптора (кроме фонетического) может быть использована опция уникального дескриптора (Unique), а для определения субдескрипора, супердескриптора, гипердескриптора и фонетического дескриптора используется альтернатива Define…

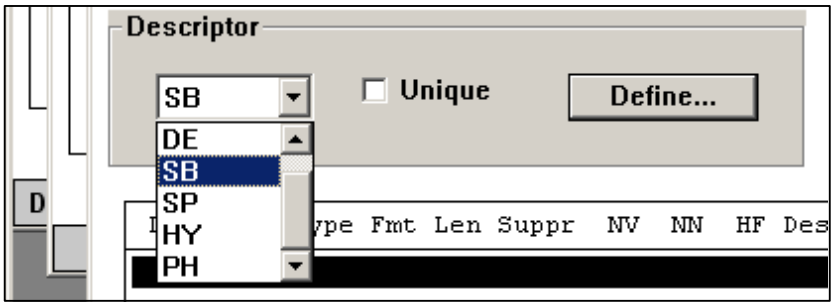


Рис. 70. Определение дескриптора

SB — описание субдескриптора

Субдескриптор — это дескриптор, созданный из части элементарного поля. Элементарное поле может быть или не быть дескриптором непосредственно. Субдескриптор может также использоваться как субполе, т. е. он может быть определен в форматном буфере для определения формата выходных записей.

Субдескриптор должен быть определен со следующими параметрами:

имя — правила присвоения имени субдескриптору идентичны тем, которые используются для имени поля Adabas;

UQ (опция Unique) — указывает, что субдескриптор должен быть определен как уникальный;

исходное поле — имя исходного поля, из которого будет получен субдескриптор;

начало — относительная позиция байта в пределах исходного поля, где начинается описание субдескриптора;

конец — относительная позиция байта в пределах исходного поля, где кончается описание субдескриптора.

Подсчет параметров «начало» и «конец» делается слева направо начиная с 1 для алфавитно-цифровых полей и справа налево начиная с 1 для цифровых и двоичных полей. Если исходное поле определено с форматом P, знак результирующего значения субдескриптора будет взят из 4 битов младшего разряда байта младшего разряда (1 байт).

Исходное поле может быть:

– дескриптором;

– элементарным полем;

– множественным полем (но не конкретная реализация множественного поля);

– входить в состав периодической группы (но не конкретная реализация периодической труппы).

Исходное поле не может быть:

– суб/суперполем, субдескриптором, супердескриптором или фонетическим дескриптором;

– G формата (с плавающей точкой).

Если поле определено с опцией NU, никакие входы не генерируются в инвертированном списке субдескриптора для тех записей, которые содержат нулевое значение (в пределах определенных байт позиций) для поля. Формат тот же самый, что и для исходного поля.

Если исходное поле не инициализировано и логически падает мимо конца физической записи, вход инвертированного списка для этой записи не формируется по причине производительности. В этом случае для создания входа инвертированного списка необходимо файл разгрузить с опцией SHORT, разуплотнить и загрузить повторно или использовать прикладную программу для инициализации поля каждой записи файла.

Синтаксис субдескриптора должен быть описан посредством использования альтернативы ‘Define’ (рис. 71).

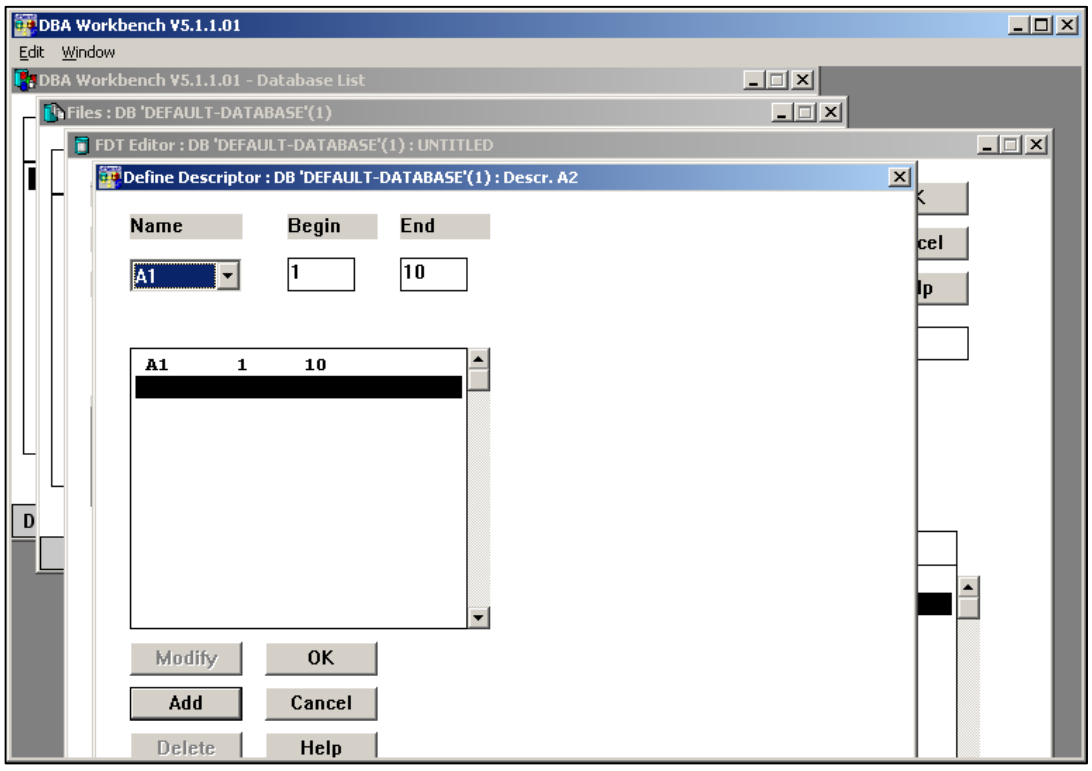


Рис. 71. Определение синтаксиса субдескриптора

В окне «Define Descriptor» необходимо определить исходное поле, начало и конец субдескриптора:

– имя исходного поля (Name). Посредством поля со списком ‘Name’ (отображаются уже определенные поля) необходимо выбрать исходное поле для субдескриптора;

– начало (Begin). В поле «Begin» указывается параметр «начало субдескриптора».

– конец (End). В поле «End» указывается параметр «конец субдескриптора»;

– альтернатива «добавить» (Add). После определения параметров имени исходного поля, начала и конца для определения субдескриптора необходимо выбрать альтернативу «Add». При этом запись определенного субдескриптора отображается в соответствующем поле;

– альтернатива «изменить» (Modify). Для изменения указанных параметров субдескриптора следует использовать опцию «Modify»;

– альтернатива «удалить» (Delete). Для удаления ранее указанных параметров субдескриптора следует использовать опцию «Delete».

Для сохранения параметров субдескриптора нужно воспользоваться системной кнопкой «OK». При определении субдескриптора следует учитывать, что в качестве параметра «исходное поле», как следует из свойств субдескриптора, может быть выбрано только одно поле из уже определенных. В противном случае значение данного параметра не будет принято системой, о чем проектировщик БД будет уведомлен сообщением (рис. 72).

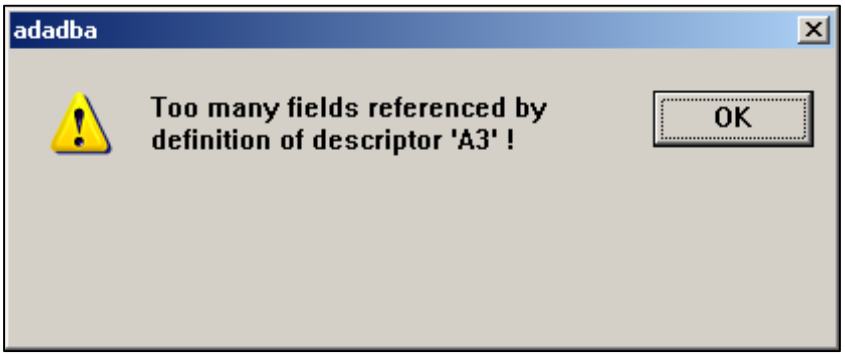


Рис. 72. Информационное сообщение,
получаемое при выборе нескольких исходных полей

Определение параметров «имя» и UQ (Unique) субдескриптора осуществляется при помощи основного окна редактора FDT.

SP — описание супердескриптора

Супердескриптор — это дескриптор, созданный из нескольких полей, частей полей или их комбинации. Каждое полепервоисточник (или часть поля) используемое для определения супердескриптора, называется исходным. Супердескриптор может быть определен с использованием от 2 до 20 исходных полей или частей поля. Супердескриптор может быть определен как уникальный дескриптор. Супердескриптор может также использоваться как суперполе, т. е. он может быть определен в форматном буфере для определения формата выходной записи.

Суперскриптор должен быть определен со следующими параметрами:

– имя — правила присвоения имени супердескриптору идентичны тем, которые используются для имени поля Adabas;

– UQ (опция Unique) — указывает, что супердескриптор должен быть определен как уникальный;

– исходное поле — имя исходного поля, из которого будет получен элемент супердескриптора; может быть определено до 20 исходных полей;

– начало — относительная позиция байта в пределах исходного поля, где начинается описание супердескриптора;

– конец — относительная позиция байта в пределах исходного поля, где заканчивается описание супердескриптора.

Подсчет параметров «начало» и «конец» делается слева направо начиная с 1 для алфавитно-цифровых полей и справа налево начиная с 1 для цифровых и двоичных полей. Для любого исходного поля, кроме тех, которые определены как FI, значения начала и конца имеют силу в пределах разрешенного диапазона для типа данных исходного поля.

Исходное поле может быть:

– элементарным или максимумом одним множественным полем (но не определенного значения множественного поля);

– в пределах периодической группы (но не определенная реализация);

– дескриптором.

Исходное поле не может быть:

– супер-, суб- или фонетическим дескриптором;

– G формата (с плавающей точкой);

– поле опции NC, если другое исходное поле — поле опции NU;

– длинным алфавитно-цифровым (LA) полем.

Если исходное поле определено с опцией NU, никакие входы не генерируются в инвертированном списке супердескриптора для записей, содержащих нулевые значения для поля. Это действительно независимо от присутствия или отсутствия значений для других элементов супердескриптора.

Если исходное поле не инициализировано и логически падает мимо конца физической записи, вход инвертированного списка для этой записи не формируется по причине производительности. В этом случае для создания входа инвертированного списка необходимо файл разгрузить с опцией SHORT, разуплотнить и загрузить повторно или использовать прикладную программу для инициализации поля каждой записи файла.

Полная длина любого значения супердескриптора не может превышать 253 байта (алфавитно-цифровые) или 126 (двоичных) байтов. Супердескриптор имеет алфавитно-цифровой формат, если какой-нибудь элемент супердескриптора получен из алфавитно-цифрового исходного поля; иначе супердескриптор имеет двоичный формат.

Все супердескрипторы двоичного формата обрабатываются как числа без знака. Синтаксис супердескриптора должен быть описан посредством использования альтернативы «Define» аналогично описанию синтаксиса субдескриптора. При определении супердескриптора, в отличие от субдескриптора, в качестве исходных могут использоваться от 2 до 20 уже определенных в таблице FDT полей.

Определение параметров «имя» и UQ (Unique) супердескриптора осуществляется при помощи основного окна редактора FDT.

HY — описание гипердескриптора

Опция гипердескриптора позволяет сгенерировать значения дескриптора на основании алгоритма, обеспеченного пользователем.

Значения основаны на алгоритмах, закодированных в специальных пользовательских выходах гипердескриптора (от HEX01 до HEX31). Каждому гипердескриптору должен быть назначен пользовательский выход, и отдельный пользовательский выход может обрабатывать множество гипердескрипторов (рис. 73).

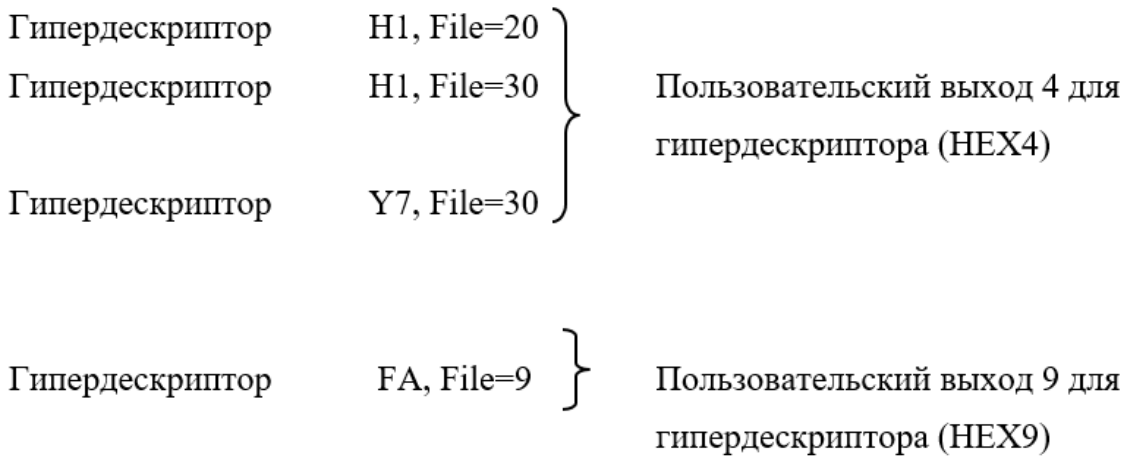


Рис. 73. Гипердескриптор и пользовательские выходы

Входные параметры необходимые для пользовательского выхода:

– имя гипердескриптора;

– номер файла;

– адреса полей, взятых из записи памяти данных, вместе с именем поля и PE индексом (если применяется). Эти адреса указывают на сжатые значения полей.

Пользовательский выход должен возвратить значение дескриптора (DVT) в сжатом формате. Может быть не возвращено никакого значения или одно или большее количество значений в зависимости от опций (PE, MU), определенных для гипердескриптора.

Исходный ISN, назначенный для входного значения, может быть изменен.

При использовании гипердескрипторов нужно учитывать следующие примечания:

– проектировщик определяет формат, длину и опции гипердескриптора. Они не могут быть взяты от исходного поля, определенного в параметре HY;

– поиск, использующий значение гипердескриптора, выполняется тем же способом, что и для стандартных дескрипторов;

– проектировщик ответствен за создание корректных значений гипердескриптора. Нет никакого стандартного пути проверки значений гипердескриптора на целостность в памяти данных. Пользователь должен установить правила описания каждого значения и проверки корректности значения;

– если формат гипердескриптора упакованный или распакованный, Adabas проверяет возвращенные значения на правильность. Знак полбайта для упакованного значения может содержать A, C, E, F (положительный) или B, D (отрицательный). Adabas преобразует знак в F или D;

– если файл содержит больше одного гипердескриптора, назначенные выходы вызываются в алфавитном порядке имен гипердескрипторов.

Гипердескриптор должен быть определен со следующими параметрами:

– номер — ядро Adabas использует этот номер для определения пользовательского выхода гипердескриптора, который будет вызван;

– имя — правила присвоения имени гипердескриптору идентичны тем, которые используются для имени поля Adabas;

– длина — по умолчанию.

– формат — форматы идентичны тем, которые используются для имени поля Adabas;

– опция — вместе с гипердескриптором могут использоваться следующие опции: MU — множественное поле; NU — подавление нулевого значения; PE — поле периодической группы; UQ — уникальный дескриптор; SV — создание поиска значений;

– исходное поле — гипердескриптор может иметь от 1 до 20 исходных полей. Имена полей и адреса передаются пользова-

тельскому выходу. Если исходное поле определено с опцией NU, никакие входы не генерируются в инвертированном списке гипердескриптора для записей, содержащих нулевые значения поля. Это не зависит от присутствия или отсутствия значений для других элементов гипердескриптора.

Если исходное поле не инициализировано и логически оно попадает мимо конца физической записи, то вход инвертированного списка для этой записи не формируется по причине производительности. В этом случае для создания входа инвертированного списка необходимо файл разгрузить с опцией SHORT, разуплотнить и загрузить повторно или использовать прикладную программу для инициализации поля каждой записи файла.

Синтаксис гипердескриптора должен быть описан посредством использования альтернативы «Define» (рис. 74).

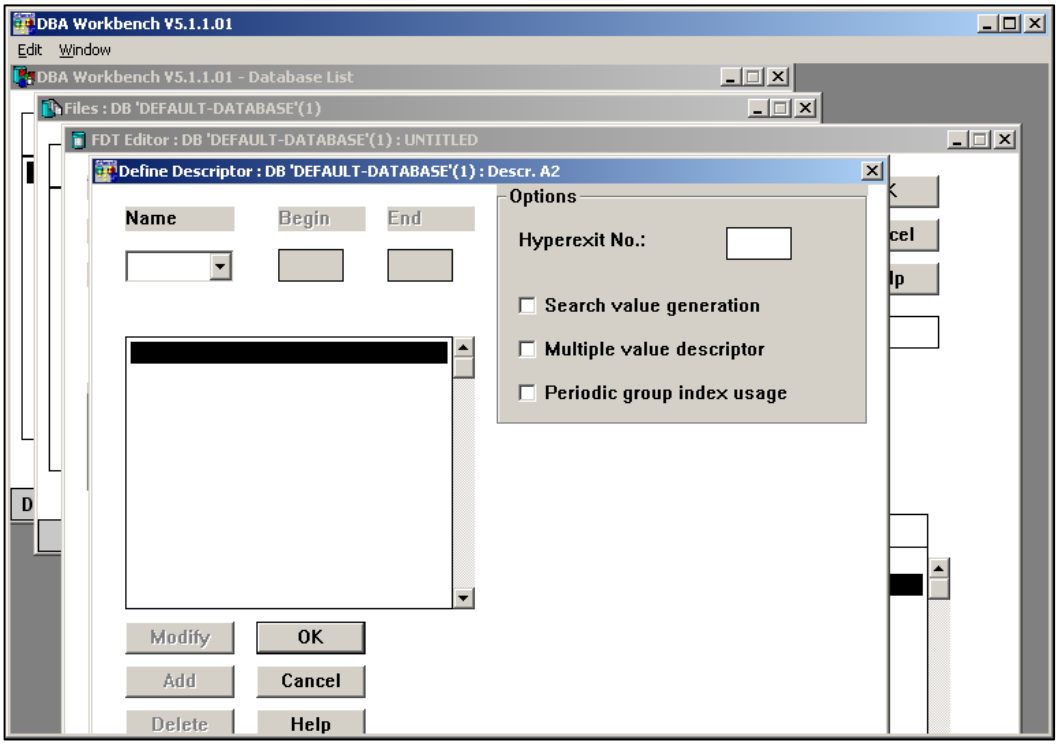


Рис. 74. Определение синтаксиса гипердескриптора

В окне «Define Descriptor» определяются следующие параметры гипердескриптора:

– имя исходного поля (Name). Посредством поля со списком ‘Name’ (отображаются уже определенные поля) необходимо выбрать исходное поле (исходные поля) для гипердескриптора;

– начало (Begin) и конец (End). В полях «Begin» и «End» автоматически устанавливается полная длина исходных полей гипердескриптора. Для проектировщика эти поля являются неактивными и не подлежат изменению;

– номер пользовательского выхода (Hyperexit No.). Посредством данного поля необходимо установить параметр «номер гипердескриптора»;

– опции (Options). Здесь определяются такие составляющие параметра «опция» как MU — альтернатива «Multiple value descriptor», PE — альтернатива «Periodic group index usage» и SV — альтернатива «Search value generation»;

– альтернатива «добавить» (Add). После определения параметров гипердескриптора, необходимо выбрать альтернативу «Add». При этом запись определенного гипердескриптора отображается в соответствующем поле;

– альтернатива «изменить» (Modify). Для изменения указанных параметров гипердескриптора следует использовать опцию «Modify»;

– альтернатива «удалить» (Delete). Для удаления ранее указанных параметров гипердескриптора следует использовать опцию «Delete».

Для сохранения параметров гипердескриптора нужно воспользоваться системной кнопкой «OK».

Определение параметров «имя», «длина», «формат» гипердескриптора, а также таких составляющих параметра «опция», как UQ и NU, осуществляется при помощи основного окна редактора FDT, аналогично определению параметров «имя», UQ (Unique) суб- и супердескриптора.

Для лучшего понимания процедуры определения гипердескриптора ниже приведен пример его описания (табл. 24).

Описание гипердескриптора:

HYPDE=’2,NH,60,A,MU,NU=LN,FN,FR’

Гипердескриптору назначен пользовательский выход 2 и имя гипердескриптора — HN.

Длина гипердескриптора — 60, формат — алфавитно-цифровой и множественное (MU) поле с подавлением нулей (NU).

Значения для гипердескриптора должны быть получены из полей LN, FN и FR.

Таблица 24

Пример описания гипердескриптора

Описание определяемых полей	Значения полей
FNDEF='01,LN,20,A,DE,NU'	Фамилия
FNDEF='01,FN,20,A,MU,NU'	Имя
FNDEF='01,ID,4,B,NU'	Идентификатор
FNDEF='01,AG,3,U'	Возраст
FNDEF='01,AD,PE'	Адрес
FNDEF='02,CI,20,A,NU'	Город
FNDEF='02,ST,20,A,NU'	Улица
FNDEF='01,FA,PE'	Родственники
FNDEF='02,NR,20,A,NU	Фамилия родственника
FNDEF='02,FR,20,A,MU,NU'	Имя родственника

PH — фонетический дескриптор

В результате выполнения команды FIND (редактор Natural) с использованием фонетического дескриптора все возвращенные записи будут содержать подобные фонетические значения (иметь аналогичное произношение). Фонетическое значение дескриптора основано на первых 20 байтах значения поля. Рассматриваются только алфавитные значения; числовые значения, специальные символы и пробелы игнорируются. Нижний и верхний регистры алфавитно-цифровых символов внутренне идентичны.

Фонетический дескриптор должен быть определен со следующими параметрами:

– имя — правила присвоения имени фонетическому дескриптору идентичны тем, которые используются для имени поля Adabas;

– поле — имя поля, которое будет транскрибировано фонетически.

Поле должно быть:

– элементарным или множественным;

– определено в алфавитно-цифровом формате.

Поле может быть дескриптором.

Поле не может:

– быть суб-, супер- или гипердескриптором;

– входить в состав периодической группы;

– использоваться как исходное поле для более чем одного фонетического дескриптора.

Если поле определено с опцией NU, никакие входы не генерируются в инвертированном списке фонетического дескриптора для тех записей, которые содержат нулевое значение (в пределах определенных байтов позиций) для поля. Формат тот же самый, что и для поля.

Если поле не инициализировано и логически падает мимо конца физической записи, вход инвертированного списка для этой записи не формируется по причине производительности. В этом случае для создания входа инвертированного списка необходимо файл разгрузить с опцией SHORT, разуплотнить и загрузить повторно или использовать прикладную программу для инициализации поля каждой записи файла.

Синтаксис фонетического дескриптора должен быть описан посредством использования альтернативы «Define» (рис. 75).

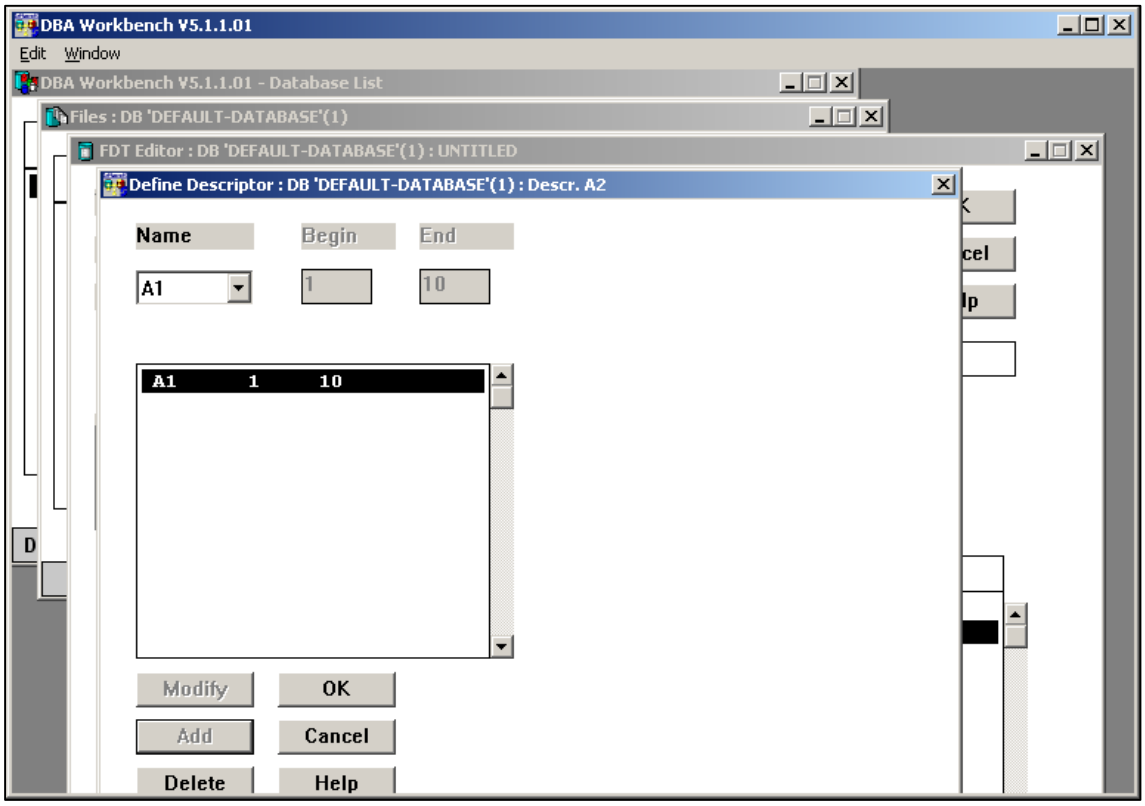


Рис. 75. Определение синтаксиса фонетического дескриптора

В окне «Define Descriptor» определяются следующие параметры фонетического дескриптора:

– имя исходного поля (Name). Посредством поля со списком «Name» (отображаются уже определенные поля) необходимо выбрать поле фонетического дескриптора;

– начало (Begin) и конец (End). В полях «Begin» и «End» автоматически устанавливается полная длина поля фонетического дескриптора. Для проектировщика эти поля являются неактивными и не подлежат изменению;

– альтернатива «добавить» (Add). После определения параметров фонетического дескриптора необходимо выбрать альтернативу «Add». При этом запись определенного фонетического дескриптора отображается в соответствующем поле;

– альтернатива «изменить» (Modify). Для изменения указанных параметров фонетического дескриптора следует использовать опцию «Modify»;

– альтернатива «удалить» (Delete). Для удаления ранее указанных параметров фонетического дескриптора следует использовать опцию «Delete».

Для сохранения параметров фонетического дескриптора нужно воспользоваться системной кнопкой «OK».

Определение параметра «имя» фонетического дескриптора осуществляется при помощи основного окна редактора FDT аналогично определению параметров «имя», UQ (Unique) суб- и супердескриптора.

Ниже приведен пример описания фонетического дескриптора:

Описание поля: FNDEF='01,AA,20,A,DE,NU'
Фонетическое описание: PHONDE='PA(AA)'

NN — SQL опция не нулевого значения

При помощи редактора FDT для каждого элементарного поля может быть задана функция NN (Not NULL).

Опция NN («не ноль» или «нулевое значение не разрешено») может быть определена, только когда опция NC также определена для поля. Опция NN указывает, что NC поле должно всегда иметь определенное значение (включая ноль или пробел); оно не может содержать «нет значения».

Опция NN гарантирует, что поле не будет оставлено неопределенным, когда запись добавлена или обновлена; поле всегда должно устанавливаться в значащее значение.

Следующий пример показывает, как незначащий ноль будет обработан в двухбайтном алфавитно-цифровом поле AA, когда оно определено с и без опции NN (табл. 25).

Таблица 25

Пример представления нулей с опцией NN

Опция	Описание поля	Значение нулевого индикатора	Внутреннее представление Adabas
С опцией NN	01,AA,2,A,NC,NN	FFFF (незначащие нули)	Нет, ошибка
Без функции NN	01,AA,2,A,NC	FFFF (незначащие нули)	C1

Ниже следует пример корректного описания поля AA с назначением опций NC и NN:

FNDEF='01,AA,4,A,NN,NC,DE'

Далее следует пример неправильного использования опций NC/NN. Эти описания приведут к тому, что в результате будет получена ошибка:

FNDEF='01,AA,4,A,NC,NU' (опции NU и NC несовместимы)
FNDEF='01,AB,4,A,NC,FI' (опции NC и FI не совместимы)
FNDEF='01,PG,PE' (не разрешается использовать опцию NC
FNDEF='02,P1,4,A,NC' внутри PE группы)

Альтернатива «добавить» (Add). После определения каждого поля следует выбрать альтернативу «Add» в основном окне редактора FDT. При этом системой осуществляется проверка синтаксиса каждого поля. Если системой будут выявлены какие-либо ошибки, проектировщик будет осведомлен посредством сообщений. Для продолжения работы с редактором необходимо исправить все ошибки, выявленные системой.

Альтернатива «изменить» (Modify). Для внесения изменений в уже введенные параметры необходимо выделить (с помощью курсора мыши) нужное поле и воспользоваться альтернативой «Modify».

Альтернатива «удалить» (Delete). В случае когда необходимо удалить какое-либо из определенных полей, следует выделить (с помощью курсора мыши) нужное поле и воспользоваться альтернативой «Delete».

После определения параметров атрибутов следует воспользоваться системной кнопкой «OK» основного окна редактора FDT и перейти к процедуре сохранения файла.

5.5. Определение параметров ассоциатора и обработки больших данных

Под определением элементов файла в рамках создания БД подразумевается определение экстентов таких параметров, как хранилище данных (Data Storage, DATA), адресный конвертор (Address Converter, AC), стандартный индекс (Normal Index, NI), верхний индекс (Upper Index, UI). Перечисленные параметры используются для указания типа и размера экстента, который будет определен. Экстенты этих параметров вводятся и корректируются при помощи опций при сохранении файла БД.

Для определения экстентов данных параметров необходимо создать файл в рамках существующей БД и определить все поля для внесения пользовательских данных. При сохранении определенного файла (при помощи окна «Create File») необходимо воспользоваться системной кнопкой «Опции» («Options…») (рис. 76).

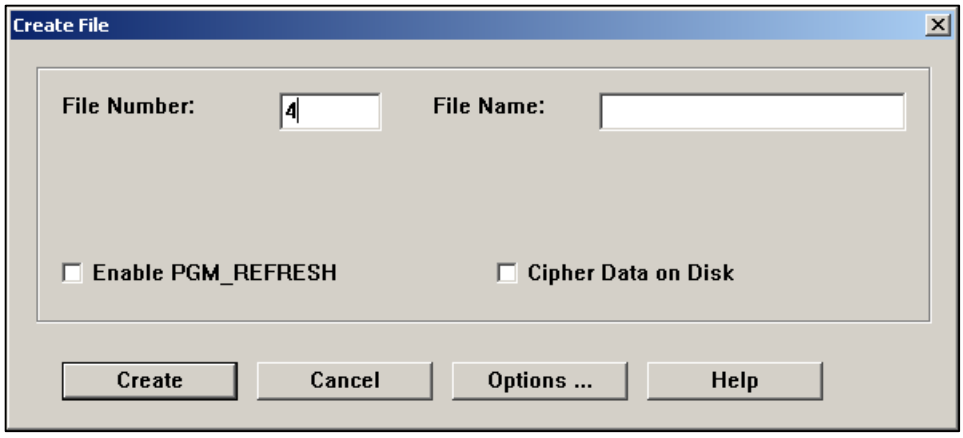


Рис. 76. Определение экстентов параметров

Далее проектировщику будет предложено окно для определения параметров (рис. 77).

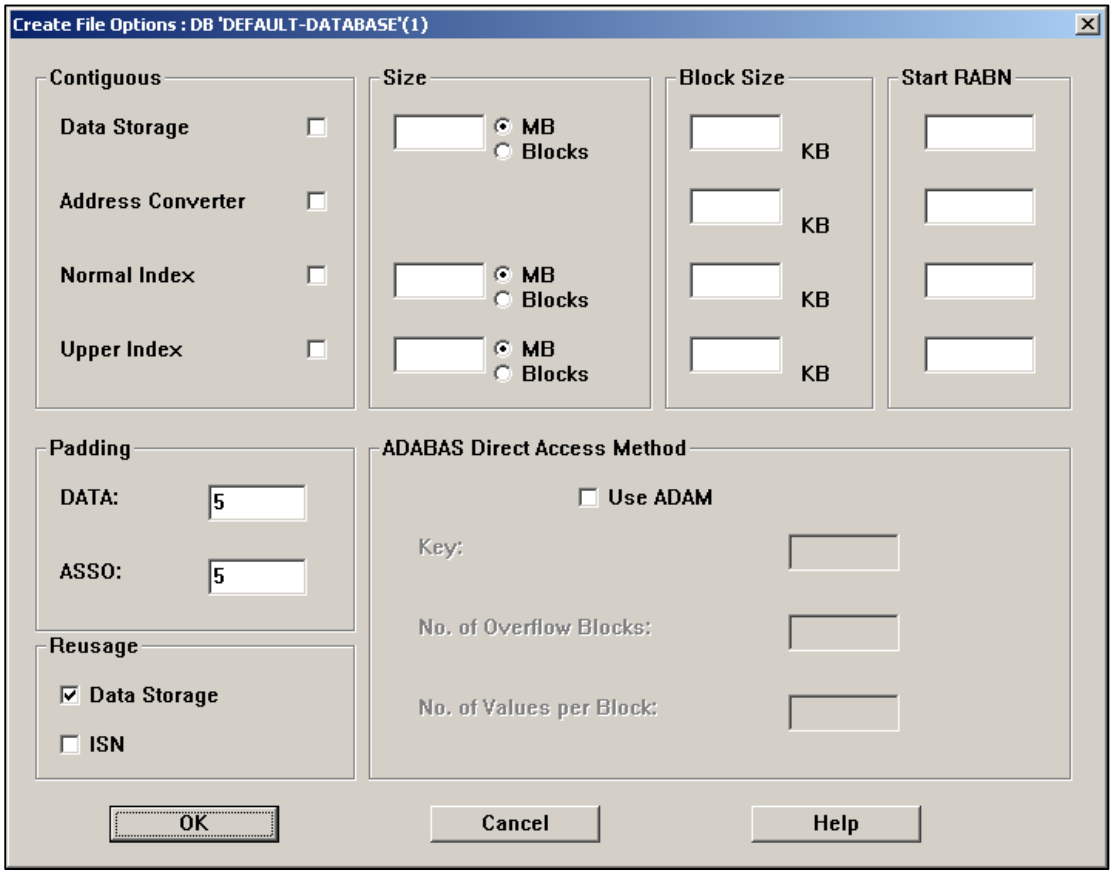


Рис. 77. Определение параметров

В появившемся окне предлагаются следующие группы для ввода параметров:

– непрерывные (Contiguous). Для внесения значений любого из параметров — хранилище данных, адресный конвертор, стандартный индекс, верхний индекс — необходимо поставить флажок напротив соответствующего параметра;

– размер (Size). В данной группе значений указывается размер (в блоках — опция «Blocks» или мегабайтах — опция «MB») каждого из параметров — хранилище данных, стандартный индекс, верхний индекс. Размер адресного конвертора фиксирован;

– размер блока (Block Size). Для определения размера блоков каждого из параметров необходимо ввести значение (в Кбайтах) в соответствующее параметру поле данной группы;

– начальный RABN (Start RABN). Если необходимо разместить область какого-либо из параметров на определенном дисковом сегменте или в специальном экстенте, следует ввести значение начального RABN в соответствующее параметру поле. В случае когда начальный RABN не определен, система размещает нижние ряды последовательных свободных RABN так, чтобы обеспечить достаточно свободного дискового пространства для каждого из элементов. Если размеры и начальный RABN не определены проектировщиком, они будут автоматически назначены для каждого из вышеперечисленных параметров;

– дополнительное пространство (Padding). При определении значений дополнительного пространства для файла ассоциатора (ASSO) и файла данных (DATA) системой резервируется область дополнительного дискового пространства для записей, которые могут потребовать дополнительное место при обновлении. Это позволяет избежать перемещения записей в иные блоки. Если ожидается незначительное (или отсутствие) обновление дескрипторов (файл ASSO) и незначительное (или отсутствие) расширение записей (файл DATA), следует установить значение данного параметра в диапазоне от 0 до 10. Если же ожидается широкое обновление дескрипторов и большое расширение записей, значение параметра необходимо определить в диапазоне от 10 до 50;

– повторное использование (Reusage). Данная опция используется для повторного использования блоков данных и номеров ISN. При использовании опции «хранение данных» (Data Storage) при добавлении новой записи или перемещении записей в результате осуществления обновлений системой используется первый найденный блок, в котором имеется достаточно пространства для размещения записи. При использовании опции «номер ISN» (ISN) номера ISN тех записей, которые были удалены, перемещаются (присваиваются иным записям), в противном случае (когда данная опция не используется) при добавлении новой записи ей присваивается следующий (наибольший) номер ISN. Особенность использования опции можно рассмотреть на примере (табл. 26).

Таблица 26

Пример использования опции «номер ISN»

Последовательность ISN		
до удаления записей и внесения новой записи	при использовании опции *Reusage номер ISN*	без использования опции *Reusage номер ISN*
…	…	…
149	149	149
150	150	152
151	151	153
152		

В первоначальном состоянии имелось несколько записей, и их номера ISN были расположены по порядку (первая колонка табл. 26). Допустим, что было удалено две записи (соответствующих номерам ISN 150 и 151) и одна запись была добавлена. Результат присвоения номеров ISN в данном случае при и без использования опции «Reusage номер ISN» показан во второй и третьей колонках табл. 26.

Метод направленного запроса Adabas (Adabas Direct Access Method, ADAM)

Метод направленного запроса Adabas (ADAM) позволяет поиск записей прямо из хранилища данных без обращения к инвертированным спискам. Номер блока хранилища данных, в котором располагается запись, высчитывается на основе рандомизированного алгоритма — ADAM-ключа записи. ADAM-ключ каждой записи должен быть уникальным. Номер ISN записи также может использоваться в качестве ADAM-ключа. Для использования опции ADAM должны быть определены следующие параметры:

– использовать ADAM (Use ADAM). При введения параметров для использования в рамках метода ADAM необходимо выбрать данную опцию. Это делает поля для ввода параметров активными для проектировщика;

– ADAM-ключ (Key). В данном поле указывается сокращенное имя (Short Name) определенного в файле дескриптора либо номер ISN. В качестве ADAM-ключа не может использоваться суб-, супер-, фонетический или гипердескриптор, а также

множественное поле или поле в рамках заданной периодической группы и поля, определенные с опцией NU/NC. В качестве ADAM-ключа необходимо указывать какой-либо из уникальных дескрипторов, определенных в рамках создаваемого файла, либо номер ISN. В случае когда объявленный ADAM-ключ не является уникальным дескриптором, проектировщик будет проинформирован сообщением (рис. 78);

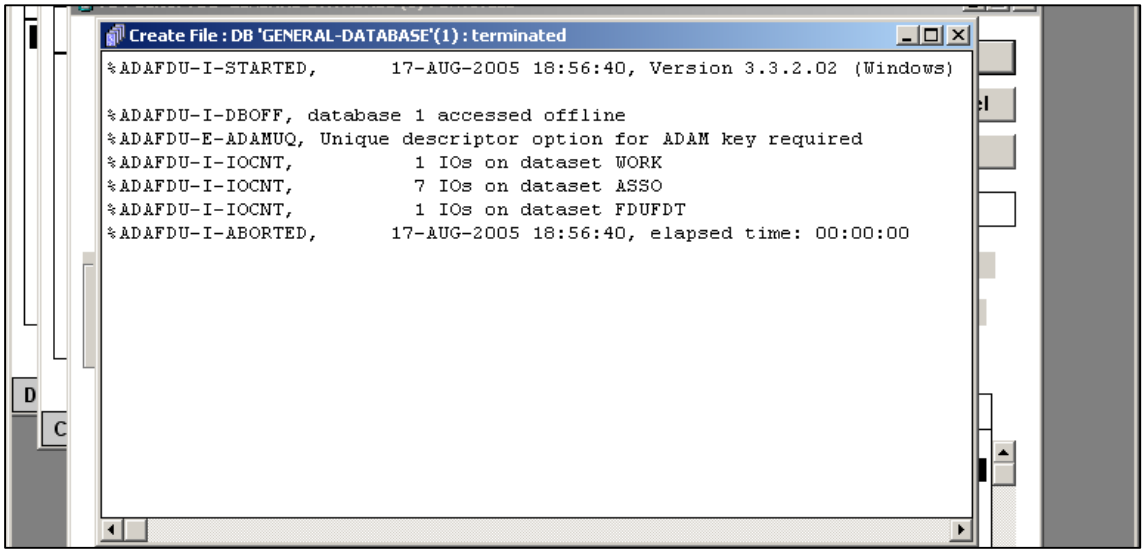


Рис. 78. Информационное сообщение, получаемое если ADAM-ключ не является дескриптором

– количество дополнительных блоков (No. Of Overflow Blocks). При использовании опции ADAM необходимо определить количество блоков, отводимых дополнительно для осуществления операций. Требуется как минимум один блок. Количество блоков может быть увеличено в дальнейшем. Дополнительные блоки используются, когда недостаточно дискового пространства для посчитанного методом ADAM блока новой записи. Преимущество при выполнении операций с использованием метода ADAM повышается с ростом количества дополнительных блоков;

– количество значений в блоке (No. of Values per Block). Данный параметр используется для распределения записей данных. Если ADAM-ключ определен как ISN, или дескриптор с фиксированной точкой, данный параметр определяет количество последовательных значений, которые будут храниться в од-

ном блоке. Если ADAM-ключ является дескриптором формата алфавитно-цифрового, двоичного или с плавающей точкой, данный параметр определяет смещение с конца значения записи на 4 байта (значение, используемое как ключ).

Следующий пример наглядно показывает действие данного параметра (рис. 79).

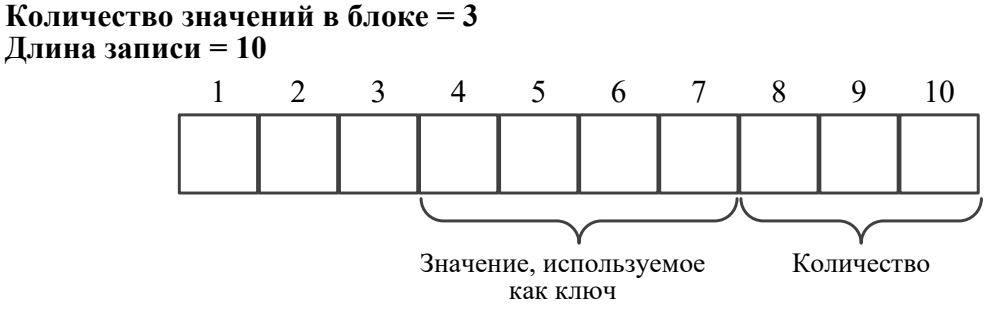


Рис. 79. Действие параметра ADAM (формата алфавитно-цифрового, двоичного или с плавающей точкой)

Если длина значения для определения смещения оказывается меньше чем 4 байта, то это значение будет расширено до нужной длины двоичными нулями.

Если формат ADAM-ключа запакованный или не запакованный, количество значений в блоке определяет смещение (в байтах) с конца значения записи до той позиции, в которой значение может быть рассмотрено в качестве значения ADAM. Это значение ADAM будет преобразовано в 4-байтовое целочисленное значение. Дальнейший пример иллюстрирует действие данного параметра для запакованного и распакованного формата ADAM-ключа (рис. 80).

Рис. 80. Действие параметра ADAM (формата запакованный или не запакованный)

Главное преимущество использования дескрипторов в качестве ADAM-ключа — понижение количества запросов к списку номеров ISN. Главное преимущество использования ISN в качестве ADAM-ключа — уменьшение количества запросов к адресному конвертору.

После определения всех параметров следует воспользоваться системной кнопкой «OK» и перейти к сохранению файла.

Выводы по главе 5

Одной из первых реализаций, обладающих инструментарием для основных моделей данных нереляционных СУБД, можно считать промышленную СУБД Adabas.

База данных Adabas определяется как совокупность взаимосвязанных файлов и ассоциаций записей файлов. Файл представляет собой именованную совокупность записей, имеющих одинаковую структуру. Ассоциация записей файлов представляет собой совокупность записей файла, обладающих общим свойством.

База данных Adabas размещается на устройствах прямого доступа. Записи БД запоминаются в блоках устройств прямого доступа. Размер блока выбирается с учетом условия эффективного размещения целого числа блоков на дорожке используемых типов устройств прямого доступа.

Структурными элементами базы данных Adabas на внутреннем уровне являются: накопитель, ассоциатор, рабочий набор и вспомогательные наборы данных. Ассоциатор используется для получения системных таблиц, преобразователя адреса и инвертированных списков. Рабочий набор данных предназначен для временного размещения промежуточной информации необходимой в процессе работы с БД.

Процедура определения базы данных включает три операции: физическое размещение, создание и редактирование файлов-контейнеров базы данных.

Вопросы для самоконтроля

1. Раскройте понятия «файл» и «ассоциация». Раскройте понятия «атрибут», «группа», «периодическая группа».
2. Опишите структуру данных СУБД Adabas и организацию связей между записями файлов БД. Каким образом хранятся записи файла и как строятся инвертированные списки?
3. Как в рамках СУБД Adabas реализуется связь типа «многие-ко-многим»?
4. Назовите операции модификации схемы БД и операции модификации.
5. Охарактеризуйте операции чтения, поиска, селекции. В чем их различие? Назовите и опишите разновидности селекции.
6. Воспроизведите схему и алгоритм ассоциативного поиска с использованием инвертированных списков и алгоритм ассоциативного поиска записей в связанных файлах.
7. Назовите и опишите структурные элементы базы данных СУБД Adabas.
8. Какова структура накопителя? Как задается объем накопителя и можно ли его увеличить в процессе эксплуатации?
9. Опишите структуру ассоциатора СУБД Adabas. Что такое таблица FDT и в каком структурном элементе она хранится?
10. В каком виде хранятся инвертированные списки в ассоциаторе СУБД Adabas? Как осуществляется доступ к требуемому инвертированному списку?
11. Как происходит преобразование BCH в адрес?
12. Опишите структуру рабочего и вспомогательных наборов данных.

Практические задания

1. Создайте БД в среде СУБД Adabas с параметрами по умолчанию для ассоциатора, накопителя и рабочего набора.
2. Измените размеры ассоциатора и рабочего набора на предпочтительные значения. Измените порядковые номера системных файлов. Загрузите демофайлы в БД.
3. В рамках созданной БД создайте новый файл. При помощи таблицы FDT опишите структуру хранения данных о студентах (включая порядковый номер студента, Ф.И.О. студента, название института, номер группы, социальный статус, семейное положение, результаты

сессии по 8 дисциплинам, размер стипендии). Порядковый номер студента объявите дескриптором.

4. Создайте новый файл, в котором при помощи таблицы FDT, используя периодические группы, создайте структуру хранения данных для 5 групп студентов (данные о студентах см. в п. 3). Используйте функцию «фиксированное значение» для номера группы. Сохраните созданный файл с опциями шифрования данных на диске, размером накопителя 200 Мб, размером блока 2 Кб и включением метода направленного запроса.

Заключение

Мы рассмотрели эволюцию подходов к хранению и обработке данных, основные понятия и проблемы в отношении больших данных. Подробно разобрали модели данных начиная от самых первых традиционных сетевой и иерархической модели, заканчивая современными нереляционными системами типа «ключ-значение» и «большая таблица».

В заключение необходимо отметить, что все рассмотренные модели имеют право на существование и могут достаточно эффективно использоваться в конкретных случаях. При выборе той или иной модели данных следует руководствоваться масштабами проектируемой системы, территориальным охватом, количеством пользователей, функциональным предназначением системы и специфическими особенностями выполняемых задач. Спектр технологий обработки больших данных велик, и выбор той или иной технологии также обусловлен спецификой задачи, которую требуется решить в конкретных условиях, в конкретное время и обладая определенными ограниченными ресурсами.

Отрасль больших данных, несомненно, и дальше будет развиваться быстрыми темпами. Одним из наиболее актуальных направлений является применение облачных технологий при обработке больших данных промышленных предприятий с целью построения цифровых двойников. Также актуальным и перспективным является развитие машинного обучения и интеллектуального анализа больших данных.

Для дальнейшего освоения дисциплины авторы рекомендуют изучить основы технологий машинного обучения и их реализации на языке Python.

Библиографический список

Использованная литература

1. *Информационные* технологии управления: учеб. пособие / В. П. Часовских, Г. А. Акчурина, А. В. Слободин [и др.]. 4-е изд., испр. и доп. Екатеринбург: Урал. гос. лесотехн. ун-т, 2015. 667 с.

2. *Черпаков И. В.* Теоретические основы информатики: учебник и практикум для акад. бакалавриата. М.: Юрайт, 2017. 353 с.

3. *Майер-Шенбергер В., Кукьер К.* Большие данные. Революция, которая изменит то, как мы живем, работаем и мыслим / пер. с англ. И. Гайдюк. М.: Манн, Иванов и Фербер, 2014. 240 с.

4. *Тиндал С.* Большие данные: все, что вам необходимо знать // PC Week/RE. 2012. № 25 (810). URL: https://www.it-week.ru/idea/article/detail.php?ID=141962.

5. *Zaiane O. R.* Principles of Knowledge Discovery in Databases. 1999. Chapter I: Introduction to Data Mining. URL: https://webdocs.cs.ualberta.ca/~zaiane/courses/cmput690/notes/Chapter1/index.html.

6. *Селезнев К.* Проблемы анализа Больших Данных // Открытые системы. СУБД. 2012. № 07. URL: https://www.osp.ru/os/2012/07/13017638.

7. *Чехарин Е. Е.* Большие данные: большие проблемы // Перспективы Науки и Образования. 2016. № 3 (21). С. 7–11.

8. *Вайгенд А.* BIG DATA. Вся технология в одной книге. М.: Эксмо, 2018. 384 с.

9. *Интеллектуальный* анализ больших пространственно-временных данных для служб экстренного реагирования / А. Р. Гараева, Р. Н. Минниханов, М. В. Дагаева [и др.] // Совре-

менные информационные технологии и ИТ-образование. 2018. Т. 14, № 3. С. 679–685.

10. *Силен Д., Мейсман А., Али М.* Основы Data Science и Big Data. Python и наука о данных. СПб.: Питер, 2017. 336 с.

11. *Власов А. И., Лыткин С. Л., Яковлев В. Л.* Краткое практическое руководство разработчика информационных систем на базе СУБД Oracle: Библиотечка журнала «Информационные технологии». М.: Машиностроение, 2000. 120 с.

12. *Воронов М. П., Фатеркин А. С., Часовских В. П.* Информационные технологии в управлении: СУБД ADABAS и проектирование приложений средствами NATURAL. Екатеринбург: Урал. гос. лесотехн. ун-т, 2006. 477 с.

13. *Зеленков Ю. А.* Введение в базы данных. 1997. URL: http://www.mstu.edu.ru/study/materials/zelenkov/ch_6_2.html.

14. *Селезнев К.* От SQL к NoSQL и обратно // Открытые системы. СУБД. 2012. № 02. URL: https://www.osp.ru/os/2012/02/13014127

15. *Кузнецов С. Д., Посконин А. В.* Системы управления данными категории NoSQL // Программирование. 2014. Т. 40, № 6. С. 34–47.

16. *What* is a Key-Value Database? Database.Guide. URL: https://database.guide/what-is-a-key-value-database/.

17. *What* is a Document Store Database? Database.Guide. URL: https://database.guide/what-is-a-document-store-database/.

18. *What* is a Column Store Database? Database.Guide. URL: https://database.guide/what-is-a-column-store-database/.

19. *Redis.* Official web site. Introduction to Redis. URL: https://redis.io/topics/introduction.

20. *Project* Voldemort. Official web site. URL: http://www.project-voldemort.com/voldemort/.

21. *Aerospike.* Official web site. Architecture Overview. URL: https://www.aerospike.com/docs/architecture/index.html.

22. *MongoDB.* Official web site. Introduction to MongoDB. URL: https://docs.mongodb.com/manual/introduction/.

23. *OrientDB.* Official web site. Overview. URL: https://orientdb.org/docs/3.0.x/misc/Overview.html.

24. *OrientDB.* Official web site. Data modeling. URL: https://orientdb.org/docs/3.0.x/datamodeling/Tutorial-Document-and-graph-model.html.

25. *Apache* Cassandra. Official web site. Architecture. Overview. URL: https://cassandra.apache.org/doc/latest/architecture/overview.html.

26. *Apache* Druid. Official web site. Introduction to Apache Druid. URL: https://druid.apache.org/docs/latest/design/index.html.

27. *Apache* Druid. Official web site. Design. URL: https://druid.apache.org/docs/latest/design/architecture.html.

28. *Егерев С. В., Захарова С. А.* Краудсорсинг в науке // Наука. Инновации. Образование. 2013. № 14. С. 175–186.

29. *Митева Ц.* Краудсорсинг в бизнесе: коллективный разум спешит на помощь. URL: https://kontur.ru/articles/415.

30. *Волошинская А. А.* Краудсорсинг: новая классификация для анализа новых возможностей // Информационное общество. 2015. № 4. С. 31–38.

31. *Sexton D. G., Oh O., Kishor R.* Rules of Crowdsourcing: Models, Issues, and Systems of Control // Information Systems Management. 2013. Vol. 30, issue 1.

32. *Инновационная* экономика для современного мира / авт. кол.: В. А. Балашов, И. Я. Львович, А. Б. Почтовюк [и др.]. Одесса: КУПРИЕНКО СВ, 2018. 118 с.

33. *Меморандум* I Международного научного форума «Шаг в будущее: искусственный интеллект и цифровая экономика» (Москва, 6–7 декабря 2018 г.) // Муниципальная академия. 2018. № 4. С. 2–6.

34. *Сигель Э.* Просчитать будущее. Кто кликнет, купит, соврет или умрет. М.: Альпина Паблишер, 2018. 374 с.

35. *Бердышев А. В.* Искусственный интеллект как технологическая основа развития банков // Вестник университета. 2018. № 5. С. 91–94.

36. *Дулёв А. А.* Внедрение искусственного интеллекта в деятельность кредитных организаций // Хроноэкономика. 2018. № 5 (13). С. 27–30.

37. *Малыгина Л. Е.* Чат-боты и искусственный интеллект: перспективы развития телевизионного промодискурса // Акту-

альные проблемы филологии и педагогической лингвистики. 2018. № 4 (32). С. 47–54.

38. *Шапошников К. С., Сагаева И. Д., Сидоров С. П.* Анализ сложных сетевых структур на примере социальной сети «Twitter» // Информационные технологии и математическое моделирование (ИТММ-2019): материалы XVIII Междунар. конф. им. А. Ф. Терпугова (Саратов, 26–30 июня 2019 г.). Томск: Изд-во научно-технической литературы, 2019. Ч. 2. С. 71–74.

39. *Кузнецов Е. Н.* Анализ структуры сетевых взаимодействий: контекстно-зависимые меры центральности // Управление большими системами: сб. тр. 2019. № 80. С. 57–82.

40. *Bonacich P.* Power and Centrality: A family of Measures // American Journal of Sociology. 1987. № 5 (92). С. 1170–1182.

41. *Bonacich P., Lloyd P.* Eigenvector-like measures of centrality for asymmetric relations // Social Networks. 2001. № 3 (23). С. 191–201.

42. *Системы* управления базами данных ДИСОД / Е. С. Броневщук, В. И. Бурдаков, Л. И. Гуков [и др.]; под общ. рук. В. И. Дракина. М.: Финансы и статистика, 1987. 263 с.

Рекомендуемая литература

Ын Анналин, Су Кеннет. Теоретический минимум по Big Data. Всё, что нужно знать о больших данных. СПб.: Питер, 2019. 208 с.

Андреас Вайгенд. BIG DATA. Вся технология в одной книге. М.: Эксмо, 2018. 384 с.

Силен Д., Мейсман А., Али М. Основы Data Science и Big Data. Python и наука о данных. СПб: Питер, 2017. 336 с.

Варнавский А. В., Бурякова А. О., Себеченко Е. В. Блокчейн на службе государства. М: КноРус, 2020. 215 с.

Информация об авторах

Часовских Виктор Петрович — профессор кафедры шахматного искусства и компьютерной математики УрГЭУ, доктор технических наук, профессор

e-mail: u2007u@ya.ru

Воронов Михаил Петрович — доцент кафедры шахматного искусства и компьютерной математики УрГЭУ, кандидат технических наук, доцент

e-mail: mstrk@yandex.ru

Лабунец Валерий Григорьевич — профессор кафедры шахматного искусства и компьютерной математики УрГЭУ, доктор технических наук, профессор

e-mail: vlabunets05@yahoo.com

Стариков Евгений Николаевич — заведующий кафедрой шахматного искусства и компьютерной математики УрГЭУ, кандидат экономических наук, доцент

e-mail: starikov_en@usue.ru

Иванов Игорь Владимирович — старший преподаватель кафедры шахматного искусства и компьютерной математики УрГЭУ

e-mail: igor_v_ivanov@mail.ru

Оглавление

Учебное издание

Часовских Виктор Петрович,
Воронов Михаил Петрович,
Лабунец Валерий Григорьевич
и др.

Формализация информации и big data

Учебное пособие

Редактор и корректор *Л. В. Матвеева*
Компьютерная верстка *И. В. Засухиной*

Поз. 23. Подписано в печать 18.08.2021.
Формат 60 × 84 1/16. Бумага офсетная. Печать плоская.
Уч.-изд. л. 9,0. Усл. печ. л. 12,8. Печ. л. 13,8. Заказ 412. Тираж 81 экз.
Издательство Уральского государственного экономического университета
620144, г. Екатеринбург, ул. 8 Марта/Народной Воли, 62/45

Отпечатано с готового оригинал-макета в подразделении оперативной полиграфии
Уральского государственного экономического университета