МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Е.И. ЧИГАРИНА

БАЗЫ ДАННЫХ

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский государственный аэрокосмический университет имени академика С.П. Королева (национальный исследовательский университет)» в качестве учебного пособия для студентов, обучающихся по направлению подготовки бакалавров 230100.62 Информатика и вычислительная техника

С А М А Р А
Издательство СГАУ
2015

УДК 004(075)
ББК 32.97я7
Ч 586

Рецензенты: д-р техн. наук, проф.С.А. П и я в с к и й ;
канд. техн. наук, доц. Л.А. Ж а р и н о в а ;
канд. пед. наук, доц. М.В. Д о д о н о в

Чигарина Е.И.

Ч586 Базы данных: учеб. пособие / *Е.И. Чигарина*. – Самара: Изд-во СГАУ, 2015. – 208 с.

ISBN 978-5-7883-1031-2

Изложены основные концепции баз данных, а также практическая ориентация на разработку локальных и распределенных баз данных с применением современных СУБД, используемых в автоматизированных информационных системах оперативной обработки данных.

Учебное пособие предназначено для студентов – бакалавров, обучающихся по очной форме по направлению 230100.62 Информатика и вычислительная техника по курсу «Базы данных», и подготовлено на кафедре информационных систем и технологий Самарского аэрокосмического университета.

УДК 004(075)
ББК 32.97я7

ISBN 978-5-7883-1031-2

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Понятие баз данных. Виды баз данных. Средства реализации баз данных. Основные этапы создания баз данных.

Одной из самых распространенных сфер применения вычислительной техники является создание различных автоматизированных систем. Среди них САПР (системы автоматизированного проектирования), АСУ (автоматизированные системы управления), АИС (автоматизированные информационные системы), АСНИ (автоматизированные системы научных исследований) и другие. Главное отличие этих систем – вид автоматизируемой человеческой деятельности. В САПР осуществляется автоматизация проектирования, в АСУ – управления (причем системы автоматизированного управления технологическими процессами называются АСУ ТП, управления производством – АСУП), в АИС – хранения и поиска данных, в АСНИ – научно-экспериментальных исследований. В отличие от автоматических систем, в автоматизированных системах часть функций выполняет человек и чаще всего это функции принятия решения.

Каждая из автоматизированных систем согласно государственным стандартам включает несколько основных видов обеспечений, в том числе техническое обеспечение, программное, *информационное,* организационное и правовое. Целью рассмотрения данного пособия является информационное обеспечение автоматизированных систем. Информационное обеспечение может быть организовано различным способом и включать данные, хранящиеся в оперативной и (или) внешней памяти. Использование баз данных в настоящее время является основным способом организации информационного обеспечения автоматизированных систем в случае необходимости долговременного хранения данных. В переводе с латинского языка данное (datum) – это факт. Тем не менее, данные не всегда соответствуют конкретным, реально существующим фактам. Иногда они описывают идею или неточное действие, поэтому в общем случае *данные* – это сведения

об объектах окружающего мира. В случае необходимости долговременного хранения данных, то есть использования не только оперативной, но и внешней памяти, используется понятие файла. *Файл* – это поименованный набор данных во внешней памяти. Работа с файлами осуществляется средствами операционной системы (просмотр, копирование, переименование, удаление, защита, создание и другие операции). Каждая операционная система поддерживает определенную файловую систему.

Несмотря на большое разнообразие определений понятия баз данных, все эти определения имеют три основные характеристики – долговременное хранение данных, данные имеют определенную структуру, данные логически взаимосвязаны и относятся к определенной предметной области. Таким образом, *База данных* – это совокупность взаимосвязанных данных, хранящихся во внешней памяти, описывающих некоторую предметную область. *Предметная область* – это часть реального мира, рассматриваемого в автоматизированной системе.

В зависимости *от способа хранения* данных различают следующие виды баз данных:

1. *Локальная база данных (централизованная) – база данных*, хранящая данные в памяти одной вычислительной машины.

2. *Распределенная база данных – база данных*, хранящая данные в памяти различных ЭВМ вычислительной сети.

В зависимости *от вида хранимой* информации различают следующие виды баз данных:

1. *Фактографические базы данных* – базы данных, хранящие информацию в виде данных, отражающих фактические значения, или иначе текущее состояние предметной области.

2. *Динамические базы данны*х – базы данных, хранящие данные и время их внесения или изменения, отображающие состояние предметной области в определенный момент времени.

3. *Документальные базы данных* – базы данных, хранящие информацию в виде документов, то есть в виде определенным образом организованной информации, включая отчеты, монографии и другие виды документов.

4. *Графические базы данных* – базы данных, хранящие информацию в виде графических объектов, например видеоданные, “puctoral date base”, “graphics based data base” и другие.

5. *Интегрированные* базы данных – базы данных, хранящие информацию в виде данных, документов, графических объектов в любой комбинации.

В зависимости *от структуры хранимых данных или от модели логического уровня представления данных* различают следующие виды баз данных: иерархические, сетевые, реляционные и др. Более подробно особенности этих баз данных будут рассмотрены в первой главе пособия.

Важным вопросом при работе с базами данных является вопрос выбора средства реализации баз данных. Можно выделить следующие *виды основных средств реализации баз данных*:

1. Разработчик базы данных, используя возможности файловой системы, создает на языке высокого уровня собственный сервис пользователя, позволяющий работать с файлами и реализующий определенные методы доступа к данным. Такие средства обычно более экономичны с точки зрения необходимой памяти на реализацию систем баз данных, но требуют больших затрат на разработку и теряют свои преимущества по сравнению с другими средствами реализации баз данных в универсальности.

2. Система управления базами данных (СУБД) – средство реализации баз данных, выполняющее все функции файловой системы, функции создания и ведения базы данных как совокупности взаимосвязанных файлов, функции манипулирования данными. К функциям файловых систем относятся возможности создания, удаления, переименования, изменения структуры файлов баз данных. К функции *ведения* данных относятся операции добавления, удаления, изменения, просмотра данных. К функции *манипулирования* данными относятся операции сортировки данных, поиска, выборки, фильтрации данных.

3. Машины баз данных – программно-аппаратные средства реализации баз данных. Машины баз данных не стали таким же массовым инструментом разработчика автоматизированных информационных систем, как СУБД. Это связано с тем, что машины баз данных, имея преимущество в быстродействии выполнения многих функций баз

данных, по сравнению с СУБД, так как многие функции реализуются аппаратно, тем не менее проигрывают в универсальности применения обычных персональных компьютеров. Машины баз данных позволяют преодолеть те ограничения производительности СУБД, которые вызваны сложностью их архитектуры и выполняемых ими функций управления данными. Решение этой проблемы особенно актуально для создания систем распределенной обработки данных. Новое оборудование повышает уровень надежности систем баз данных, позволяет разгрузить универсальные ЭВМ и за счет этого обеспечить более эффективное функционирование прикладных систем. В дальнейшем предполагается использование машин баз данных не только в качестве периферийного оборудования, но и в качестве самостоятельных процессоров – узлов сети ЭВМ. Использование машин баз данных будет способствовать прогрессу распределенных баз данных и систем баз знаний, а также прогрессу в работе с базами данных больших объемов. Следует отметить, что в настоящее время наиболее распространенным средством реализации баз данных являются СУБД.

Жизненный цикл создания баз данных включает этапы проектирования, реализации и эксплуатации. На этапе проектирования базы данных создается ее структура. На этапе реализации базы данных выполняется материализация проекта с помощью одного из средств реализации баз данных. На этапе эксплуатации выполняется наполнение проекта конкретной информацией и ее обновление.

К основным *этапам проектирования баз данных* относятся следующие этапы:

1. Анализ предметной области и определение требований к базе данных.
2. Концептуальное (инфологическое) проектирование базы данных.
3. Выбор средства реализации базы данных.
4. Логическое (датологическое) проектирование базы данных.
5. Физическое проектирование базы данных.

При определении требований к базе данных выполняется оценка качественных и количественных критериев. К качественным критериям относятся возможность восстановления базы данных после сбоев, возможность расширения или изменения структуры базы данных, на-

личие средств защиты информации, обеспечение целостности данных и другие критерии. К количественным критериям относятся ограничения на объем данных, ограничения на время отклика на запросы пользователей, стоимость хранения данных, стоимость обновления и другие критерии.

Задача инфологического (концептуального) этапа проектирования баз данных – построение семантических (смысловых) моделей данных.

Задача логического этапа проектирования баз данных – организация данных в виде структур данных или выбор и построение модели данных логического уровня представления данных.

Задача физического этапа проектирования баз данных – выбор рациональной структуры хранения данных и методов доступа к ним, исходя из того арсенала методов и средств, которые представляются разработчику после выбора средства реализации базы данных и этапа логического проектирования.

В современных информационных технологиях встречается не только понятие базы данных, но и базы знаний, банка данных, систем баз данных.

База знаний – это не только совокупность данных, но и совокупность знаний и набора правил вывода новых знаний. Аналогом СУБД для баз данных является понятие экспертной системы для баз знаний. *Экспертные системы* – это управляющая система, интерпретирующая правила вывода при работе с базами знаний.

Банк данных – это информационная система, реализующая централизованное управление данными. В состав банка данных входят база данных, СУБД, словарь данных, вычислительные системы, администратор базы данных. Таким образом, банк данных – более широкое понятие по сравнению с базами данных. Иногда ставят тождество между понятием банка данных и понятием автоматизированной информационной системы. Банк данных, как и любая автоматизированная система, имеет пять основных видов обеспечений: программное обеспечение, информационное, техническое, организационно-методическое и правовое.

Система баз данных – автоматизированная система хранения и обработки данных, то есть фактически это определение автоматизи-

рованной информационной системы и банка данных. В настоящее время системы баз данных делят на системы оперативной (иногда называют операционной) обработки данных или системы OLTP (OnLine Transaction Processing), а также системы аналитической обработки данных или OLAP (OnLine Analytikal Processing). Цели этих систем и модели представления данных в таких системах существенно отличаются. В данном учебном пособии будут рассмотрены системы баз данных OLTP.

Глава 1. МОДЕЛИ ДАННЫХ

1.1. Уровни представления данных. Понятия схемы и подсхемы данных

При разработке баз данных в зависимости *от степени абстрагирования данных* выделяют три уровня представления данных: концептуальный уровень, логический и физический.

Концептуальный уровень – это семантический (смысловой) уровень представления данных в виде абстрактных понятий, учитывающих особенности предметной области. На данном уровне вводятся абстрактные понятия, такие как сущность и связь, агрегация и обобщение. Названия этих понятий для разных моделей концептуального уровня представления данных отличаются, но в целом они определяют особенности объектов предметной области и взаимосвязи между ними.

Логический уровень – уровень представления данных в виде структуры данных, к которым относятся иерархические, сетевые структуры и другие виды структур, используемые при организации данных в вычислительных системах.

Физический уровень – уровень представления данных, учитывающий способ организации данных на машинном носителе в виде бит, байт и их структур. Как видно, уровень абстракции данных уменьшается от концептуального уровня представления данных к физическому уровню.

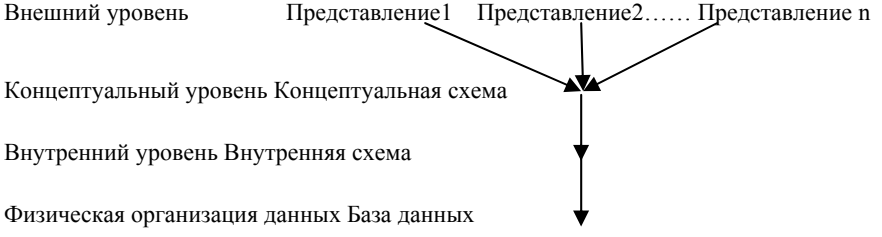
Основные этапы проектирования баз данных (концептуальное, логическое, физическое проектирование) также соответствуют трехуровневой архитектуре представления данных.

Следует отметить, что на западе принята двухуровневая система представления данных, используемая при описании баз данных. Рассматриваются логический и физический уровни представ-

ления данных. Это связано с тем, что в настоящее время при разработке баз данных в основном используются реляционные или объектно-реляционные модели данных и этапы концептуального и логического проектирования реляционных баз данных совмещены. Отсюда все современные CASE-средства проектирования баз данных позволяют строить логические и физические модели баз данных (см. раздел 1.5).

На каждом уровне представления данных имеются различные модели представления данных. Фактически синонимом понятия модели данных является понятие схемы базы данных.

Схема базы данных – описание базы данных. По аналогии с уровнями представления данных различают три типа схем баз данных в зависимости от уровня абстракции трехуровневой архитектуры.



При описании базы данных возможно наличие нескольких внешних схем или подсхем, соответствующих представлениям о данных предметной области различными пользователями. На концептуальном уровне описание базы данных называют концептуальной схемой, на нижнем уровне абстракции – внутренней схемой. Следует различать описание базы данных и саму базу данных. Описанием базы данных является схема базы данных или модель данных. Схема создается в процессе проектирования базы данных. Совокупность информации, хранящаяся в базе данных в определенный момент времени, называется состоянием. Таким образом, одной и той же схеме данных может соответствовать несколько состояний базы данных. Схема базы данных иногда называется содержанием базы данных, а ее состояние детализацией. Схема базы данных показывает логическую организацию всей базы данных в целом, а *подсхема* – описание части базы дан-

ных, описание представления о данных отдельного пользователя или приложения.

1.2. Модели концептуального уровня представления данных

На концептуальном уровне представления данных используются различные модели данных, в частности модель сущность-связь и семантическая иерархическая модель.

Наибольшее распространение получила семантическая модель данных типа «сущность – связь» (модель Entity – Relation, ER – модель, модель Чена), предложенная впервые Ченом. Основными абстрактными понятиями этой модели являются понятия сущности, экземпляра сущности, атрибута и связи.

Сущность – реальный или мыслимый объект предметной области. Сущность характеризуется свойствами – *атрибутами*. Каждая сущность состоит из множества *экземпляров сущности*. Например, сущность «студент» имеет экземпляры сущности – сведения о конкретных студентах. Фамилия и номер зачётной книжки студента – это характеристики или атрибуты сущности «студент».

Между сущностями устанавливаются связи. *Связь* – соответствие или отображение между элементами двух или более множеств (между экземплярами сущностей). Различают следующие виды или типы связей:

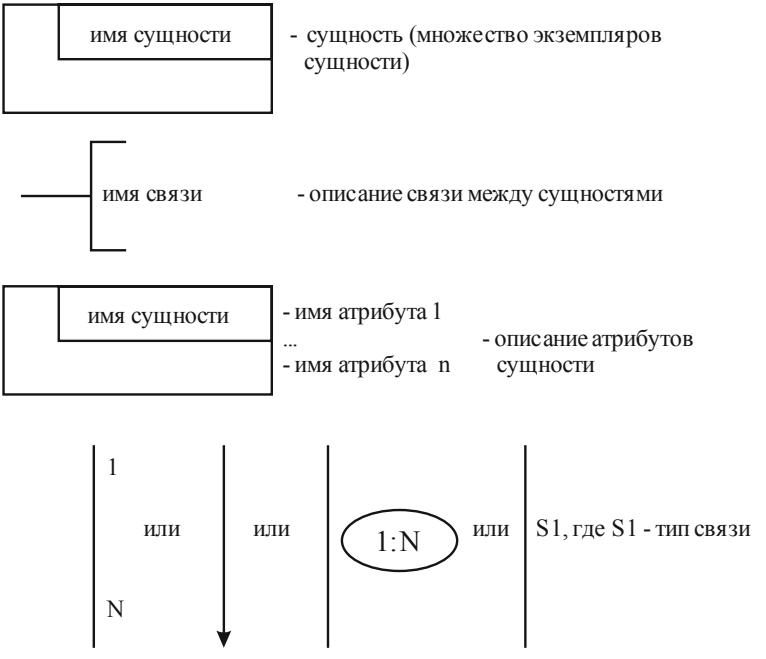
1:1 – связь один к одному между сущностями. Связь «один к одному» между сущностями устанавливается, если каждому экземпляру одной сущности соответствует только один экземпляр другой сущности.

*1:M (*или *M:1*) – связь один ко многим (или многие к одному). Связь «один ко многим» между сущностями устанавливается, если каждому экземпляру одной сущности соответствует несколько экземпляров другой сущности.

M: N – связь многие ко многим. Связь «многие ко многим» между сущностями устанавливается, если каждому экземпляру одной сущности соответствует несколько экземпляров другой сущности и наоборот.

Для описания или представления модели данных кроме введенных понятий необходимо использование некоторых графических примитивов, позволяющих эту модель показать. Существует множество вариантов графического представления модели сущность – связь. Наиболее известными являются диаграммы Бахмана и Чена.

На диаграммах Бахмана для описания модели сущность-связь используются следующие графические примитивы:



Например, дано описание следующей предметной области: в городе имеется несколько предприятий, на каждом из которых работают сотрудники. Сотрудники характеризуются должностью, фамилией, именем, отчеством (ФИО), окладом, возрастом, адресом. Предприятие характеризуется названием, числом сотрудников, ФИО директора. Каждый сотрудник может работать на одном предприятии, на одном предприятии работают несколько сотрудников.

Описание модели средствами диаграммы Бахмана представлено на рисунке 1.1.

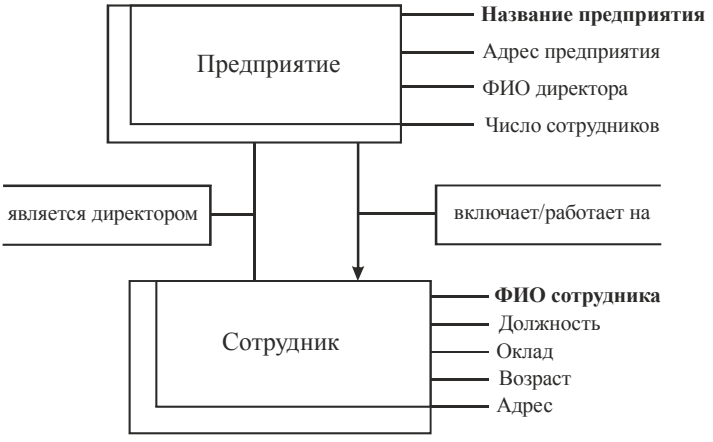
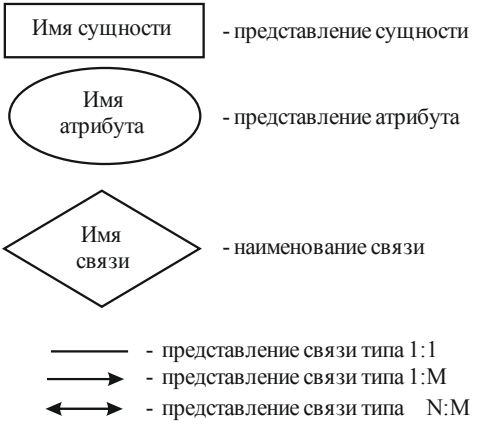


Рисунок 1.1 – Пример диаграммы Бахмана модели сущность-связь

На диаграмме Бахмана подчеркнутые атрибуты имеют неповторяющиеся значения, то есть однозначно определяют каждый экземпляр сущности. В следующих разделах будет введено понятие ключевого атрибута, соответствующего выделенным атрибутам. На рассмотренном примере показано применение графических примитивов и с точки зрения теории баз данных не учтены ограничения целостности данных, которые привели бы в частности к выделению атрибута «Должность» в отдельную сущность, вызвали бы сомнение по поводу выбора ФИО в качестве ключевого атрибута. Кроме этого, из примера видно, что между двумя сущностями может быть более одной связи. Наличие связи «один к одному» с именем «является директором» позволит при работе с базой данных быстро получить информацию о директоре более подробную, чем просто значение атрибута «ФИО директора». Кроме этого, с точки зрения логики описанной предметной области связь «один к одному» с именем «является директором» подчеркивает, что директор предприятия является сотрудником предприятия. При описании модели имеет значение направление связи между сущностями. В связи с этим при проведении связи часто вводят понятие «родительской» и «дочерней сущности». Для связи «один к одному» с именем «является директором» родительской является сущность «Сотрудник», а дочерней «Предприятие», тогда с точки зрения логики модели связь читается

как «сотрудник является директором предприятия». Имя связи на модели может читаться не только от родительской сущности по направлению к дочерней сущности, но и наоборот. Так, на рисунке 1.1 связь «один ко многим» от родительской сущности «Предприятие» имеет имя «включает», а прочтение этой же связи от дочерней сущности «Сотрудник» выглядит как «работает на». Полностью эту связь можно назвать как «сотрудник работает на предприятии». При определении имени связи в последнем случае на диаграмме имена связей записываются через «слеш» или вертикальную черту. Кроме направления, связи между сущностями имеют мощность, которая отражается в типе связи. Мощность также определяется из описания предметной области. Так, на предприятии может быть один директор и на предприятии работают несколько сотрудников. С другой стороны, только один сотрудник может быть директором и по условию сотрудник может работать на одном предприятии. Как видно из приведенного очень простого примера, задача построения модели базы данных не тривиальная, требующая знания не только правил описания модели, но и теоретических основ баз данных, существенно влияющих на эффективность работы с созданной базой данных. Эти аспекты будут рассмотрены в следующих разделах учебного пособия.

Альтернативой диаграмм Бахмана, используемых при построении модели сущность-связь, являются диаграммы Чена, которые исторически появились первыми. На диаграммах Чена используются следующие графические примитивы.



Рассмотренный ранее пример на диаграмме Чена представлен на рисунке 1.2.

С точки зрения занимаемого пространства при описании модели сущность-связь диаграмма Чена менее предпочтительна и более громоздка, тем не менее, также как и диаграмма Бахмана описывает логику взаимосвязей сущностей предметной области.

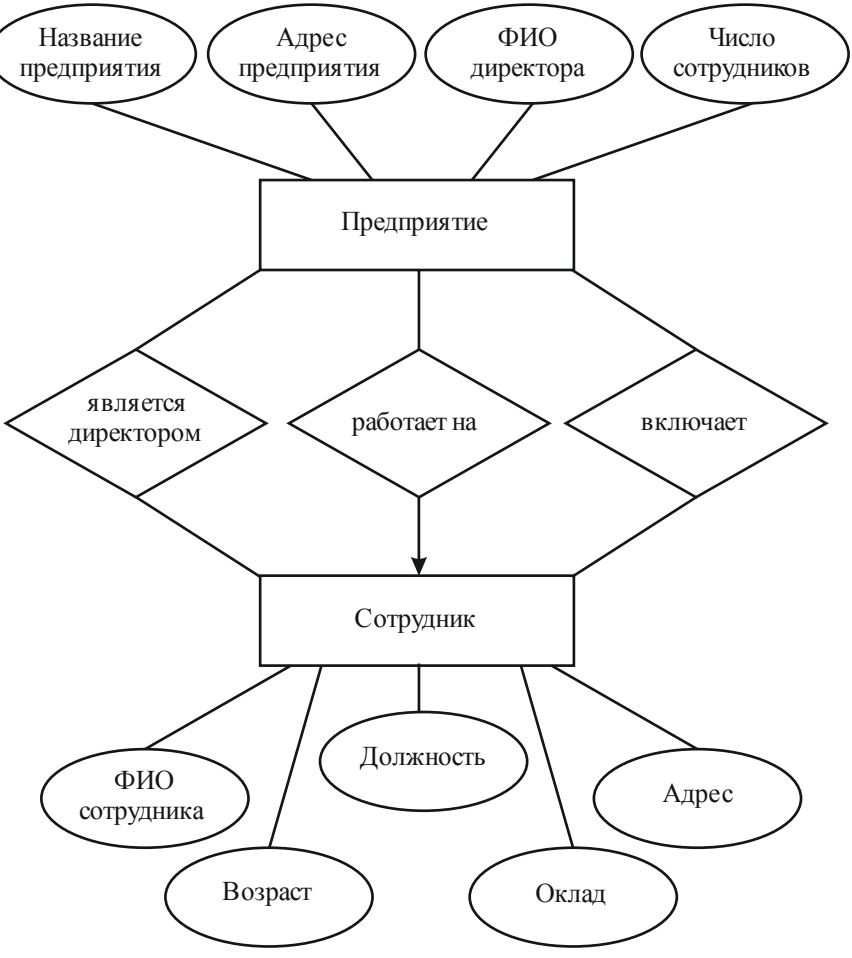


Рисунок 1.2 – Пример диаграммы Чена модели сущность-связь

Ту или иную форму графического представления модели сущность-связь выбирает проектировщик базы данных, но обязательно на любой диаграмме модели должны присутствовать сущности, атрибуты, при этом ключевые атрибуты выделены, показаны связи с указанием имен и мощностей. В настоящее время кроме приведенных диаграмм для описания моделей сущность-связь используются методологии, используемые в CASE-средствах проектирования баз данных, рассмотренные в разделах 1.4 и 1.5 пособия.

Альтернативой модели сущность-связь, используемой на концептуальном уровне представления данных, является модель Смита или семантическая иерархическая модель (SHM-модель, Semantik Hierarchic Model). Подход Смита предполагает использование множества понятий (концепций, абстракций), которые должны быть интегрированы в иерархическую структуру. Каждое из таких понятий имеет отличное от других значение, смысл которого понятен проектировщику.

В качестве элементарных форм абстракции Смит использует обобщение и агрегацию. *Обобщение* – объединение однородных элементов или объединение классов однородных элементов. *Агрегация* – объединение разнородных понятий. Например, объект автомобиль характеризуется маркой, номером, цветом кузова. Здесь можно сказать, что автомобиль агрегирует разнородные понятия – марку, цвет, номер, которые не имеют между собой ничего общего, кроме того, что они относятся к характеристикам автомобиля. В случае рассмотрения объекта линия можно сказать, что к линиям относятся окружность, прямая, эллипс. Линия имеет свойства, присущие и прямой, и окружности, и эллипсу, в то же время последние понятия имеют свои собственные, отличные от других свойства. То есть можно сказать, что понятие линия обобщает такие геометрические фигуры, как окружность, эллипс и другие.

В математическом смысле агрегация – это декартово произведение. Объекты формируются как связь между другими объектами, то есть – это объединение других объектов в единое целое по связи между ними. У каждого из класса объектов, входящих в обобщение, есть общие свойства, присущие объекту обобщения, но могут быть и

индивидуальные отличия. В обобщении наблюдается наследование свойств.

Агрегация и обобщение не являются взаимоисключающими категориями. Агрегатный объект может быть обобщением некоторого класса объектов и наоборот. Но может быть такая ситуация, когда некоторые компоненты объекта или его категории не представляют интереса и тогда они не отображаются в концептуальной модели. Объект называется *первичным*, если ни компоненты, ни категории объекта не представляют интереса. Если несколько раз последовательно применить к некоторым объектам обобщение или агрегацию, образуется иерархия объектов. В SHM-модели различают иерархии агрегаций и обобщений. В иерархии агрегаций все связи имеют тип 1:1. На рисунке 1.3 приведен пример иерархии обобщения.

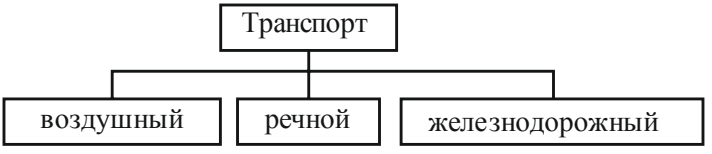


Рисунок 1.3 – Пример иерархии обобщения в SHM-модели

На рисунке 1.4 приведен пример иерархии агрегации.

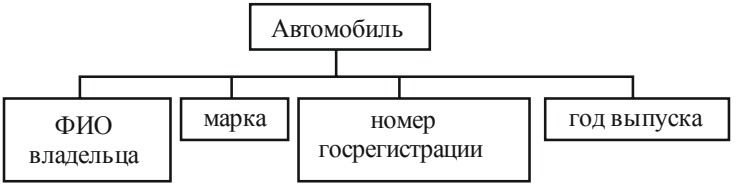


Рисунок 1.4 – Пример иерархии агрегации в SHM-модели

Иерархия абстракций является объединением иерархии обобщений и агрегаций, концептуальная модель может включать иерархии агрегации и обобщения для всех понятий прикладной области. Иерархии агрегаций и обобщений могут быть определены отдельно одна от другой, но можно определить структуру каждого объекта как единого

блока в виде объединения иерархий агрегаций и обобщения. В этом случае агрегации размещаются в плоскости страницы, а обобщенные понятия – в перпендикулярной плоскости или вводятся обозначения на связи. Связь типа ребра обобщения называется «является» и обозначается ISA; ребро иерархии агрегации – «является частью» помечается PART OF. На рисунке 1.5 приведен пример иерархии абстракций.

По сравнению с моделью сущность-связь SHM–модель более наглядна, так как иерархия всегда более ясна по сравнению с сетью или графом. Но не каждая предметная область может быть описана с помощью одной иерархии, для связей «многие ко многим» приходится строить несколько иерархий. В настоящее время при проектировании баз данных в основном используются модели сущность-связь.

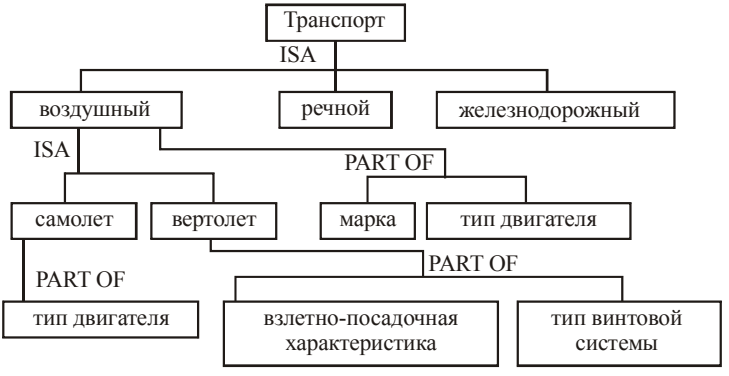


Рисунок 1.5 – Пример иерархии абстракций в SHM модели

1.3. Модели данных логического уровня представления данных

На логическом уровне данные представляются в виде некоторых структур, к которым относятся древовидные структуры, графы, таблицы, объекты. К моделям данных логического уровня представления данных относятся иерархическая модель, сетевая, реляционная, постреляционная, объектно-ориентированная. Рассмотрим основные особенности названных моделей.

В случае применения *иерархической модели данных* предполагается, что предметная область может быть разбита на отдельные части, которые можно именовать, и между данными устанавливаются иерархические отношения. Иерархическая модель реализуется в виде дерева, в узлах которого могут находиться сущности, экземпляры сущностей, атрибуты и значения атрибутов. Древовидная структура обязательно имеет корень дерева, узлы-предки и узлы-потомки. Узлы дерева, не уточняемые на более низких уровнях иерархии, называются концевыми узлами дерева. Иерархическая модель данных и семантическая иерархическая модель данных существенно отличаются. В SHM-модели присутствует иерархия понятий, а в иерархической модели логического уровня представления данных в узлах дерева могут быть экземпляры понятий и значения атрибутов. На рисунке 1.6 приведен пример иерархической модели данных.

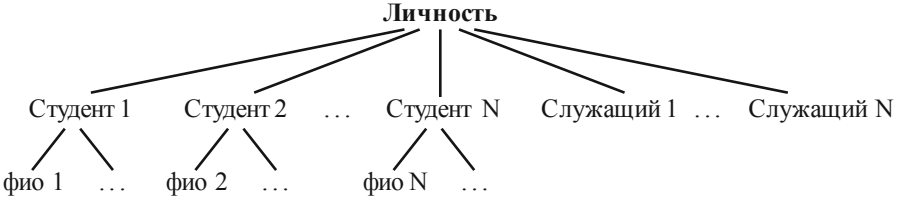


Рисунок 1.6 – Пример иерархической модели данных

Реализация связей типа «многие ко многим» на иерархических структурах приводит к необходимости дублирования информации. Например, для реализации такой связи между S и A, показанной на рисунке 1.7, надо хранить два варианта связи, иначе при поиске данных может быть выбран только один из вариантов, а не все.

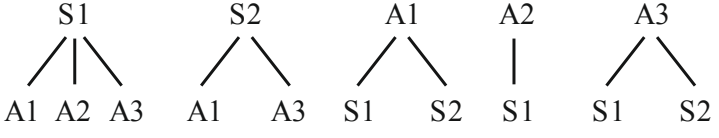


Рисунок 1.7 – Дублирование информации при реализации связей «многие ко многим» на иерархической модели

Достоинством иерархической модели является тот факт, что существуют хорошо развитые алгоритмические средства реализации древовидных структур, а также существенным фактором является наглядность и ясность иерархических структур. К недостаткам иерархической модели следует отнести избыточность при реализации связей типа «многие ко многим» и проблемы при манипулировании данными. Так, удалив узел-предок, удаляются все узлы-потомки, а это не всегда удобно. Кроме этого, невозможно хранить порожденный узел без исходного узла, то есть нужен пустой исходный узел. Для ликвидации этих недостатков была предложена сетевая модель данных.

Примерами современных СУБД, реализующих иерархическую модель данных, являются СУБД семейства IMS и System.

Сетевая модель логического уровня представляется в виде графа, возможно имеющего циклы и петли. В узлах графа могут находиться сущности, экземпляры сущностей, атрибуты, значения атрибутов. Ребрами графа являются связи между объектами графа. Модель типа “сущность – связь” лучше всего трансформируется именно на сетевую модель данных. Наиболее развитая сетевая модель данных – это модель, созданная рабочей группой по базам данных (КОДАСИЛ). Эта модель состоит из множества сущностей, которые могут быть владельцами или членами наборов данных.

Примером СУБД, реализующей сетевую модель данных КОДАСИЛ, является СУБД db_VISTA. Эта СУБД предназначена для создания и использования базы данных сложной структуры в рамках различных программных систем, реализованных на языке “С”.

Отличительная черта сетевых моделей – высокая общность, любую модель можно представить в виде сетевой модели, в этом её главное достоинство. К недостатку модели следует отнести сложность реализации такой модели, отсутствие простых и легко реализуемых алгоритмов работы с сетевыми структурами. С целью преодоления названных недостатков была разработана реляционная модель данных, самая распространённая в настоящее время.

В *реляционной модели данных* предметная область разбивается на множества, которые могут быть связаны между собой по элементам, то есть могут быть выделены упорядоченные наборы элементов, объе-

диненные в одно множество. Предполагается, что правила или алгоритмы известны и по ним могут быть выделены эти наборы. Множество получившихся наборов называется *отношением между элементами* (релейшен). Эта модель в настоящее время строго и точно формализуема. Отношение удобно представлять в виде двумерной таблицы, которая привычна и наглядна для человека. *Реляционная база данных* представляет собой совокупность взаимосвязанных отношений. Каждое отношение в ЭВМ представляется в виде файла. На рисунке 1.8 приведено соответствие понятий, используемых в реляционных моделях и присутствующих на различных уровнях описания данных.

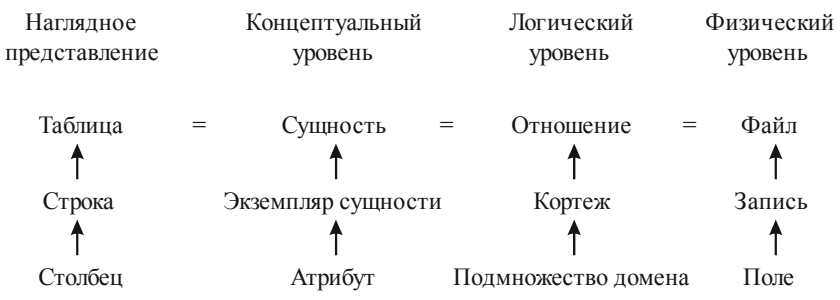


Рисунок 1.8 – Соответствие понятий реляционной модели на различных уровнях представления данных

Определение отношения, кортежа и домена будут даны в разделе логического проектирования реляционных баз данных.

Таблицы всем хороши, но не учитывают необходимости согласования трех способов манипулирования данными: упорядочение, группировку по значению индексов, доступ по дереву параметров. В таблице данные жестко связаны между собой – упорядочиваются данные по одному параметру и, следовательно, доступ по нему облегчается, а по другому параметру при этом усложняется. Но главное достоинство реляционной модели состоит в наличии строгой, формализованной теории реляционной алгебры и реляционного исчисления, лежащих в основе языков реляционных баз данных.

Примерами СУБД, поддерживающими реляционную модель данных, являются FoxPro, Clipper, Paradox и многие другие.

В отличие от реляционной модели *постреляционные базы данных* не находятся в первой нормальной форме и реализуются в виде многомерных таблиц. Постреляционные базы данных объединяют в себе высокую производительность и малую ресурсоемкость иерархических баз данных, и независимость и целостность данных, характерные для реляционных. Тем не менее, главный недостаток данных моделей – отсутствие математического аппарата их реализации.

Примером СУБД, реализующей постреляционную модель данных, является СУБД UniVerse.

В последнее время при реализации баз данных стали использоваться и объектно-ориентированные модели данных. Следуя практике многих объектно-ориентированных баз данных (ООБД), предлагается выделить два уровня моделирования объектов: нижний (структурный) и верхний (поведенческий). На структурном уровне поддерживаются сложные объекты, их идентификация и разновидности связи «isa». База данных – это набор элементов данных, связанных отношениями «входит в класс» или «является атрибутом». Таким образом, БД может рассматриваться как ориентированный граф и фактически соответствует семантической иерархической модели концептуального уровня представления данных. Важным аспектом является четкое разделение схемы базы данных и самой базы данных. В качестве первичных концепций схемного уровня ООБД (объектно-ориентированных баз данных) выступают типы и классы. Предполагается, что для точного определения ООБД требуется уровень метасхемы, содержимое которой должно определять виды объектов и связей, допустимых на схемном уровне базы данных. Метасхема должна играть для ООБД такую же роль, какую играет структурная часть реляционной модели данных для схем реляционных баз данных.

Примерами СУБД, реализующих объектную модель данных, можно назвать СУБД O2, ORION, GemStone, Iris. Распространенную СУБД ORACLE иногда называют объектно-реляционной.

1.4. Методология IDEF1X построения логических моделей реляционных баз данных

Методология IDEF1X была разработана и широко используется в государственных учреждениях США, финансовых и промышленных корпорациях при разработке реляционных баз данных. Эта методология входит в общую методологию проектирования автоматизированных систем IDEF, в основе которой лежит структурный системный анализ. В нашей стране в настоящее время методология IDEF1X, реализующая двухуровневую архитектуру представления данных, наиболее часто используется для построения логических моделей реляционных баз данных. В нотации этой методологии используется диаграмма модели сущность-связь. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области.

Сущность определяется как объект, событие или концепция, информация о которых должна сохраняться. Сущности должны иметь наименование с четким смысловым значением, именоваться существительным в единственном числе, не носить «технических» наименований и быть достаточно важным, чтобы их моделировать. *Атрибут* хранит информацию об определенном свойстве сущности, а каждый экземпляр сущности должен быть уникальным. Атрибут или группа атрибутов, которые идентифицируют сущность, называются первичным ключом. Очень важно дать атрибуту правильное имя. Атрибуты должны именоваться в единственном числе и иметь четкое смысловое значение. Соблюдение этого правила позволяет частично решить проблему нормализации данных уже на этапе определения атрибутов. *Связь* является логическим соотношением между сущностями. По отношению к каждому из перечисленных понятий, обычных для метода «сущность-связь», в методологии IDEF1X введены некоторые расширения или уточнения понятий.

В IDEF1X различают зависимые и независимые сущности. Тип сущности определяется её связью с другими сущностями. *Независимая сущность* может находиться на модели вне связи с другими сущностями и не требует дополнительных атрибутов, относящихся к другим сущностям. Независимая сущность обозначается на модели графиче-

ским примитивом в виде прямоугольника, внутрь которого включаются имена атрибутов и специально выделена область первичных ключевых атрибутов. На рисунке 1.9 приведены три варианта обозначения независимой сущности: полное атрибутивное, использующее только имена сущностей и ключевое. Понятие первичных ключевых атрибутов будет введено в этом же разделе.

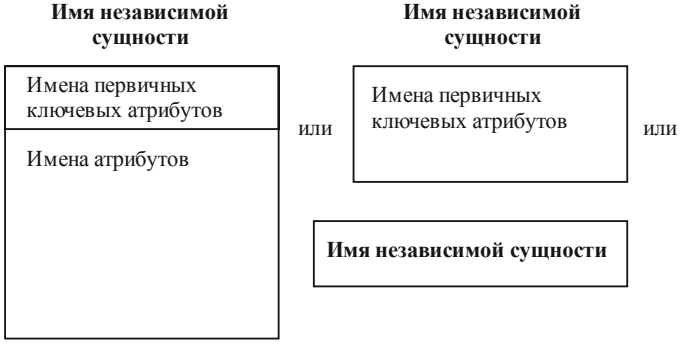


Рисунок 1.9 – Обозначения независимой сущности в методологии IDEF1X

Зависимая сущность всегда связана с другой сущностью, одной или несколькими, и обозначается прямоугольником с закругленными углами. На рисунке 1.10 приведены три варианта обозначения зависимой сущности: полное атрибутивное, использующее только имена сущностей и ключевое.

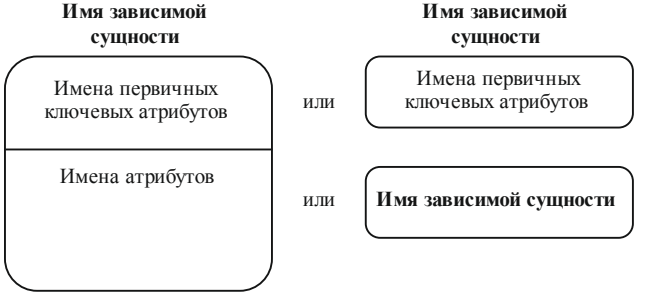


Рисунок 1.10 – Обозначения зависимой сущности в методологии IDEF1X

Выделяют несколько видов зависимых сущностей: характеристическую зависимую сущность, ассоциативную и категориальную зависимую сущности.

Характеристическая зависимая сущность – это зависимая дочерняя сущность, которая связана только с одной родительской и по смыслу хранит информацию о характеристиках родительской сущности. На рисунке 1.11 приведен пример характеристической зависимой сущности «Хобби».

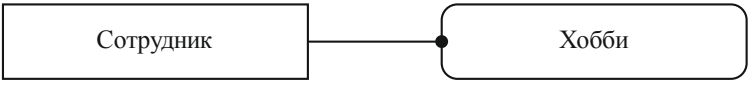


Рисунок 1.11 – Пример характеристической зависимой сущности «Хобби»

Ассоциативная зависимая сущность – сущность, связанная с несколькими родительскими сущностями. Такая сущность содержит информацию о связях сущностей. Пример ассоциативной зависимой сущности «Врач_Пациент» приведен на рисунке 1.12.

Категориальная зависимая сущность – дочерняя сущность в иерархии наследования. Пример категориальных зависимых сущностей «Постоянный сотрудник» и «Совместитель» показан на рисунке 1.17.

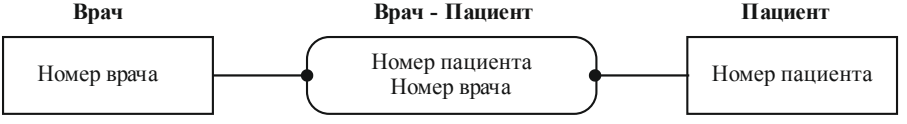


Рисунок 1.12 – Пример ассоциативной зависимой сущности «Врач–Пациент»

В нотации IDEF1X различают связи идентифицирующие, неидентифицирующие и категориальные. *Идентифицирующая связь* устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями. Экземпляр зависимой сущности определяется только через отношение к родительской сущности. Обозначается идентифицирующая связь сплошной линией с

черной точкой на конце. *Неидентифицирующая связь* устанавливается между двумя независимыми сущностями. Обозначается неидентифицирующая связь пунктирной линией с черной точкой на конце. *Категориальная связь* используется для реализации механизма наследования между сущностями. Для обозначения категориальной связи имеется специальный значок, называемый дискриминатором. На рисунке 1.13 приведены обозначения связей.

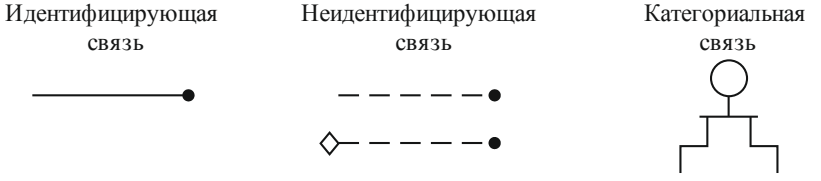


Рисунок 1.13 – Обозначения видов связей в нотации IDEF1X

При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи называется миграцией атрибутов. В дочерней сущности новые атрибуты помечаются как внешний ключ – (FK). Впоследствии, при физической генерации схемы базы данных, внешний ключ, мигрирующий в область первичных ключей, получит признак NO NULL. На рисунке 1.14 показаны пример установления идентифицирующей связи между сущностями и миграция атрибутов.

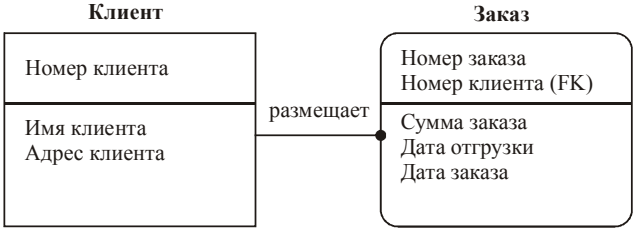


Рисунок 1.14 – Идентифицирующая связь
между зависимой и независимой сущностями

При установлении неидентифицирующей связи (рис. 1.15) дочерняя сущность остается независимой, а атрибуты первичного ключа родительской сущности мигрируют в состав неключевых компонентов родительской сущности.

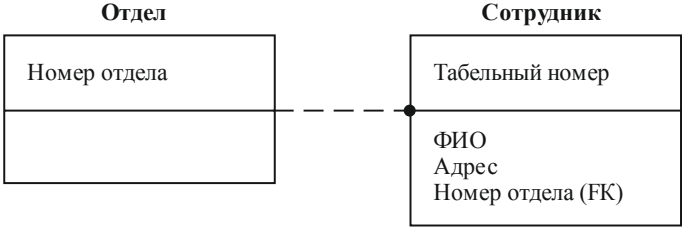


Рисунок 1.15 – Неидентифицирующая связь с признаком NOT NULL для внешнего ключа

Атрибут «Номер отдела», автоматически мигрирующий в область основных атрибутов дочерней сущности, должен быть обязательно заполнен при добавлении информации об очередном сотруднике. С точки зрения логики описания предметной области такая связь возникает в том случае, если не может быть сотрудников предприятия, работающих вне отделов, например совместителей. На рисунке 1.16 показан пример использования неидентифицирующей связи, когда допускаются сотрудники, которые не работают в отделах, присутствующих в базе данных. В этом случае мигрирующий внешний ключ имеет признак NULLS, то есть может быть пустым, не относящимся к отделам предприятия.

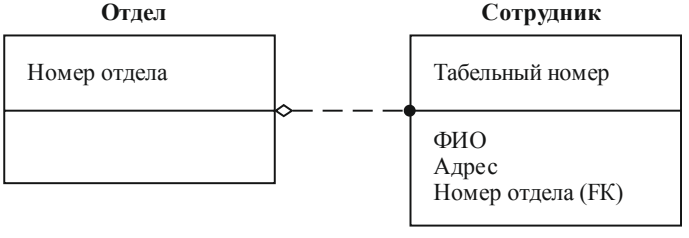


Рисунок 1.16 – Неидентифицирующая связь с признаком NULLS для внешнего ключа

Теперь рассмотрим категориальные связи, участвующие в иерархии наследования сущностей. *Иерархия наследования* (или иерархия категорий) представляет собой особый тип объединения сущностей, которые имеют общие характеристики. Например, в организации работают служащие, занятые полный рабочий день (постоянные служащие), и совместители. Из их общих свойств можно сформировать обобщенную сущность (родовой предок) «Сотрудник» (рисунок 1.17), чтобы представить информацию, общую для всех типов служащих. Специфическая для каждого типа информация может быть расположена в категориальных сущностях (потомках) «Постоянный сотрудник» и «Совместитель». Обычно иерархию наследования создают, когда сущности имеют общие по смыслу связи (например, если бы «Постоянный сотрудник» и «Совместитель» имели бы сходную по смыслу связь «работает в» с сущностью Организация). Для каждой категории указывается дискриминатор – атрибут родового предка, который показывает, как отличить одну категориальную сущность от другой.

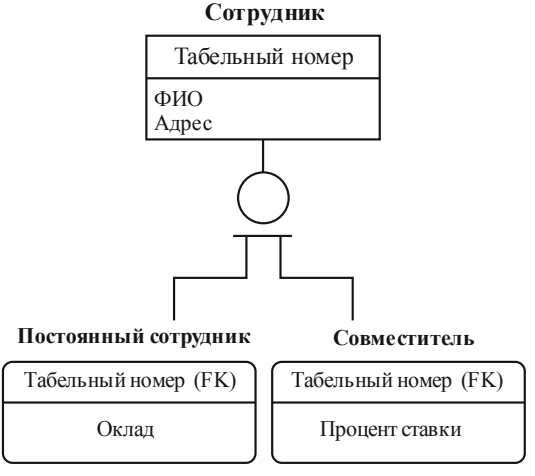


Рисунок 1.17 – Иерархия наследования. Неполная категория

Иерархии категорий делятся на два типа – полные и неполные. В полной категории одному экземпляру родового предка (сущность

«Служащий», рис.1.18) обязательно соответствует экземпляр в каком-либо потомке, то есть в примере служащий обязательно является либо совместителем, либо консультантом, либо постоянным сотрудником. Если категория еще не выстроена полностью и в родовом предке могут существовать экземпляры, которые не имеют соответствующих экземпляров в потомках, то такая категория будет неполной. На рисунке 1.17 показана неполная категория – сотрудник может быть не только постоянным или совместителем, но и консультантом, однако сущность «Консультант» еще не внесена в иерархию наследования.

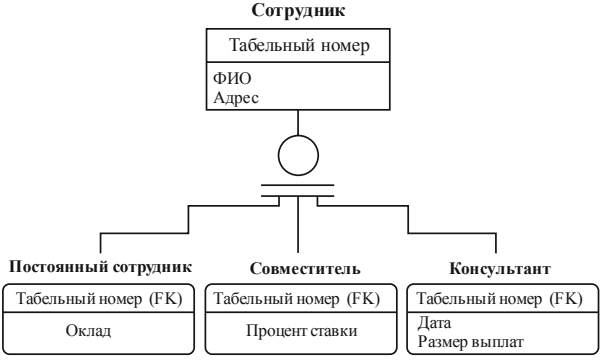


Рисунок 1.18 – Иерархия наследования. Полная категория

Возможна комбинация полной и неполной категорий. На рисунке 1.19 помимо штатных сотрудников и совместителей могут быть и консультанты, что не отражено в иерархии (неполная категория), но каждый сотрудник либо мужчина, либо женщина (полная категория).

Как уже отмечалось ранее, при рассмотрении модели сущность-связь важным понятием для любой связи является понятие мощности связи. Мощность связи (Cardinality) – служит для обозначения отношения числа экземпляра родительской сущности к числу экземпляров дочерней. В нотации IDEF1X различают четыре типа мощности (рис. 1.20):

1) общий случай, когда одному экземпляру родительской сущности соответствует 0,1 или много экземпляров дочерней сущности. На связи не помечается каким-либо символом;

2) символом P помечается случай, когда одному экземпляру родительской сущности соответствует 1 или много экземпляров дочерней сущности (исключено нулевое значение);
3) символом Z помечается случай, когда одному экземпляру родительской сущности соответствует 0 или 1 экземпляр дочерней сущности (исключены множественнные значения);
4) цифрой помечается случай точного соответствия, когда одному экземпляру родительской сущности соответствует заранее заданное число экземпляров дочерней сущности.

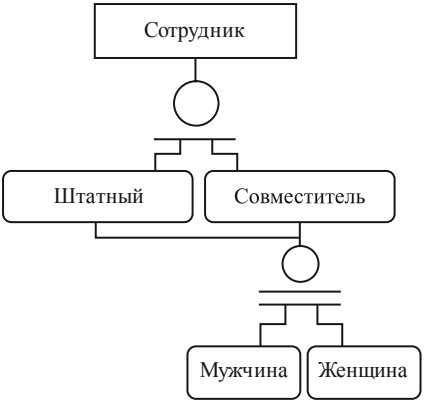


Рисунок 1.19 – Иерархия наследования.
Комбинация полной и неполной категорий

Рисунок 1.20 – Обозначения мощности

На рисунке 1.20 показано обозначение мощности для идентифицирующей связи. Аналогично, только для пунктирной линии, вводится значение мощности для неидентифицирующей связи. Мощность для категориальной связи не указывается, так как по смыслу категория связь имеет значение 0 или 1, то есть Z.

Важным для описания модели сущность-связь является указание наименования связи. Имя связи (Verb Phrase) – фраза, характеризующая отношение между родительской и дочерней сущностями. Для связи один ко многим идентифицирующей или неидентифицирующей достаточно указать имя, характеризующее отношение от родительской к дочерней (Pfrent-to-Child). Для связи многие ко многим следует указывать имена как Parent-to-Child и Child-to-Parent.

Имя роли (функциональное имя) – это синоним атрибута внешнего ключа, который показывает, какую роль играет атрибут в дочерней сущности. На рисунке 1.21 показан случай использования имени роли.

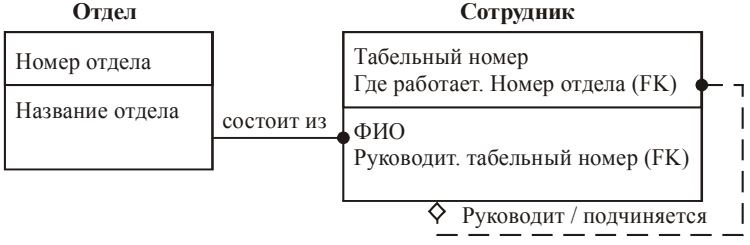


Рисунок 1.21 – Имена ролей внешних ключей

В примере, приведенном на данном рисунке, в сущности «Сотрудник» внешний ключ «Номер отдела» имеет функциональное имя «Где работает», которое показывает, какую роль играет атрибут в сущности. Обязательным является применение имен ролей в двух случаях. В случае, когда два или более атрибутов одной сущности определены по одной и той же области, то есть они имеют одинаковую область значений, но разный смысл. И второй случай был показан на рисунке 1.21, если имеется рекурсивная связь. На рисунке 1.22 показан первый обязательный случай использования имени роли для внешних ключей, когда сущности связаны более чем одной связью.

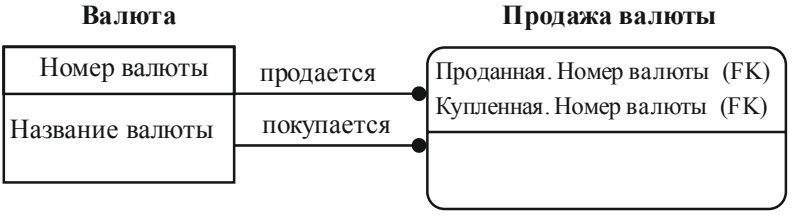


Рисунок 1.22 – Случай обязательности имен ролей

На рисунке 1.22 сущность **Продажа валюты** содержит информацию об акте обмены валюты, в котором участвуют две валюты – проданная и купленная. Информация о валютах содержится в сущности **Валюта.** Следовательно, сущности **Продажа** валюты и **Валюта** должны быть связаны дважды и первичный ключ – **Номер валюты** должен дважды мигрировать в сущность **Валюта** в качестве внешнего ключа. Необходимо различать эти атрибуты, которые содержат информацию о номере проданной и купленной валюты (имеют разный смысл), но ссылаются на одну и ту же сущность **Валюта** (имеют общую область значений). В данном примере атрибуты получили имена ролей **Проданная** и **Купленная**.

Другим примером обязательности присвоения имен ролей являются **рекурсивные связи** (иногда их называют «рыболовный крючок» – fish hook), когда одна и та же сущность является и родительской и дочерней одновременно. При задании рекурсивной связи атрибут должен мигрировать в качестве внешнего ключа в состав неключевых атрибутов той же сущности. Атрибут не может появиться дважды в одной сущности под одним именем, поэтому обязательно должен получить имя роли. На рисунке 1.21 сущность «Сотрудник» содержит атрибут первичного ключа **Табельный номер**. Информация о руководителе сотрудника содержится в той же сущности, поскольку руководитель работает в той же организации. Чтобы сослаться на руководителя сотрудника, следует создать рекурсивную связь (на рисунке связь руководит/подчиняется) и присвоить имя роли («Руководитель»). Причем рекурсивная связь может быть только неидентифицирующей. В противном случае внешний ключ должен был бы войти в состав первичного ключа и получить при генерации

схемы признак NO NULL. Это сделало бы невозможным построение иерархии – у дерева подчиненности должен быть корень – сотрудник, который никому не подчиняется в рамках данной организации.

Связь руководит/подчиняется на рисунке 1.21 позволяет хранить древовидную иерархию подчиненности сотрудников. Такой вид рекурсивной связи называется *иерархической рекурсией (hierarchical recursion)* и задает связь, когда руководитель (экземпляр родительской сущности) может иметь множество подчиненных (экземпляров дочерней сущности). Но подчиненный имеет только одного руководителя (рис. 1.23)

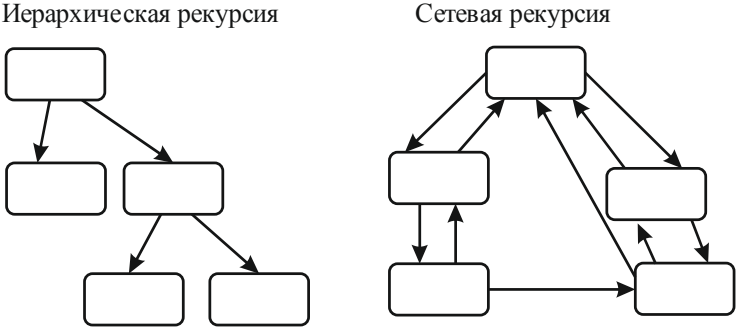


Рисунок 1.23 – Подчиненность экземпляров сущности
в иерархической и сетевой рекурсии

Другим видом рекурсии является **сетевая рекурсия (network recursion)**, когда руководитель может иметь множество подчиненных, и наоборот, подчиненный может иметь множество руководителей. Сетевая рекурсия задает паутину отношений между экземплярами родительской и дочерней сущностей. Это случай, когда сущность находится сама с собой в связи многие ко многим. Для разрешения связи многие ко многим необходимо создать новую сущность (подробно связь многие ко многим будет рассмотрена ниже). На рисунке 1.24 рассмотрен пример реализации сетевой рекурсии. Структура моделирует родственные отношения между членами семьи любой сложности. Атрибут **Тип отношения** может принимать значения «отец-сын», «мать-дочь», «дед-внук» и так далее. Поскольку родственное отноше-

ние связывает всегда двух людей, от сущности **Родственник** к сущности **Родственное отношение** установлены две идентифицирующие связи с именами ролей «Старший» и «Младший». Каждый член семьи может быть в родственных отношениях с любым другим членом семьи, более того, одну и ту же пару родственников могут связывать разные типы родственных отношений.

Рисунок 1.24 – Пример реализации сетевой рекурсии

Если атрибут мигрирует в качестве внешнего ключа более чем на один уровень, то на первом уровне отображается полное имя внешнего ключа (имя роли + базовое имя атрибута), на втором и более – только имя роли. На рисунке 1.25. изображена структура данных, которая содержит сущность **Команда**, сущность **Игрок**, в которой хранится информация об игроках каждой команды, и сущность **Гол**, содержащую информацию и о голах, которые забивает каждый игрок. Атрибут внешнего ключа **Номер команды** сущности **Игрок** имеет имя роли «В какой команде играет». На следующем уровне, в сущности **Гол**, отображается только имя роли соответствующего атрибута внешнего ключа (**В какой команде играет**).

Связь многие ко многим возможна только на уровне концептуальной модели данных. На рисунке 1.26 показан пример связи многие ко многим. Врач может принимать много пациентов, пациент может лечиться у нескольких врачей. Такая связь обозначается сплошной линией с двумя точками на концах.

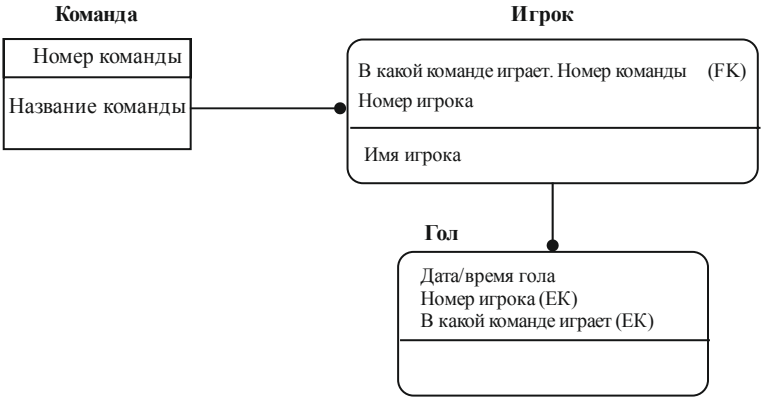


Рисунок 1.25 – Миграция имен ключей

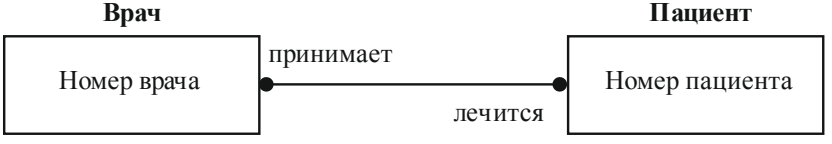


Рисунок 1.26 – Связь «многие ко многим»

Связь многие ко многим должна именоваться двумя фразами – в обе стороны (в примере «принимает/лечится»). Развязка этой связи на модели представлена на рисунке 1.12 с помощью введения дополнительной сущности «Врач-Пациент». Один и тот же пациент может много раз посещать врача, поэтому для того, чтобы идентифицировать визит, необходимо в состав первичного ключа сущности «Врач-Пациент» добавить дополнительную колонку, например дату-время посещения.

Важными понятиями в IDEF1X являются понятие ключа и различные типы ключей.

Каждый экземпляр сущности должен быть уникален и отличаться от других атрибутов.

Первичный ключ (Primary key) – это атрибут или группа атрибутов, однозначно идентифицирующих экземпляры сущности. Атри-

буты первичного ключа на диаграмме не требуют специального обозначения – это те атрибуты, которые находятся в списке атрибутов выше горизонтальной линии.

Выбор первичного ключа может оказаться непростой задачей, решение которой может повлиять на эффективность будущей информационной системы. В одной сущности могут оказаться несколько атрибутов или наборов атрибутов, претендующих на роль первичного ключа. Такие претенденты называются *потенциальными ключами (candidate key)*.

Ключи могут быть *сложными (составными)*, то есть содержащими несколько атрибутов. Сложные первичные ключи не требуют специального обозначения – это список атрибутов выше горизонтальной линии.

Например, рассмотрим кандидатов на первичный ключ сущности Сотрудник. Здесь можно выделить следующие потенциальные ключи:

- Табельный номер.
- Номер паспорта.
- Фамилия + Имя + Отчество.

Для того чтобы стать первичным, потенциальный ключ должен удовлетворять ряду требований: *Уникальность*. Два экземпляра не должны иметь одинаковых значений возможного ключа. Потенциальный ключ Фамилия+Имя+Отчество является плохим кандидатом, поскольку в организации могут работать полные тезки. *Компактность*. Сложный возможный ключ не должен содержать ни одного атрибута, удаление которого не приводило бы к утрате уникальности. Для обеспечения уникальности ключа Фамилия+Имя+Отчество дополним его атрибутами Дата рождения и Цвет волос, а ключ Фамилия+Имя+Отчество+Дата рождения+Цвет волос не является компактным.

При выборе первичного ключа предпочтение должно отдаваться более простым ключам, то есть в нашем примере это либо Табельный номер, либо номер паспорта.

Атрибуты ключа не должны содержать нулевых значений. Если допускается, что сотрудник может не иметь паспорта или вместо паспорта иметь какое-либо другое удостоверение личности, то ключ Но-

мер паспорта не подойдет на роль первичного ключа. Если для обеспечения уникальности необходимо дополнить потенциальный ключ дополнительными атрибутами, то они не должны содержать нулевых значений.

Значение атрибутов ключа не должно меняться в течение всего времени существования экземпляра сущности. Сотрудница организации может выйти замуж и сменить как фамилию, так и паспорт. Поэтому ключи Номер паспорта и Фамилия+Имя+Отчество не выбраны в качестве первичного ключа.

Каждая сущность должна иметь, по крайней мере, один потенциальный ключ. Многие сущности имеют только один потенциальный ключ. Такой ключ становится первичным. Некоторые сущности могут иметь более одного возможного или потенциального ключа. Тогда один из них становится первичным, а остальные – альтернативными ключами. *Альтернативный ключ (Alternate Key)* – это потенциальный ключ, не ставший первичным.

При работе информационной системы часто бывает необходимо обеспечить доступ к нескольким экземплярам сущности, объединенным каким-либо одним признаком. Для повышения производительности в этом случае используются неуникальные индексы. Атрибуты, участвующие в неуникальных индексах, по которым будет осуществляться выборка данных, называются *инверсионными входами (Inversion Entry)***.** *Инверсионные входы* – это атрибут или группа атрибутов, которые не определяют экземпляр сущности уникальным образом, но часто используются для обращения к экземплярам сущности.

На диаграмме атрибуты альтернативных ключей обозначаются как (AKn.m), где n – порядковый номер ключа, m – порядковый номер атрибута в ключе. Когда альтернативный ключ содержит несколько атрибутов, (AKn.m) ставится после каждого.

На рисунке 1.27 атрибуты *Фамилия, Имя, Отчество и Дата рождения* входят в альтернативный ключ № 1 (AK1), Номер паспорта составляет альтернативный ключ № 2 (AK2). Инверсионные входы обозначаются как (IEn.m), где n – порядковый номер входа, m – порядковый номер атрибута. Инверсионный вход IE1 (атрибут *Должность*) позволяет выбрать всех сотрудников, занимающих одинаковую долж-

ность, IE2 (атрибуты *Город* и *Улица*) – всех сотрудников, живущих на одной улице, IE3 (атрибут *Номер комнаты*) – всех сотрудников, работающих в одной комнате, а IE4 (атрибут *Дата рождения*) – всех сотрудников, родившихся в один день.

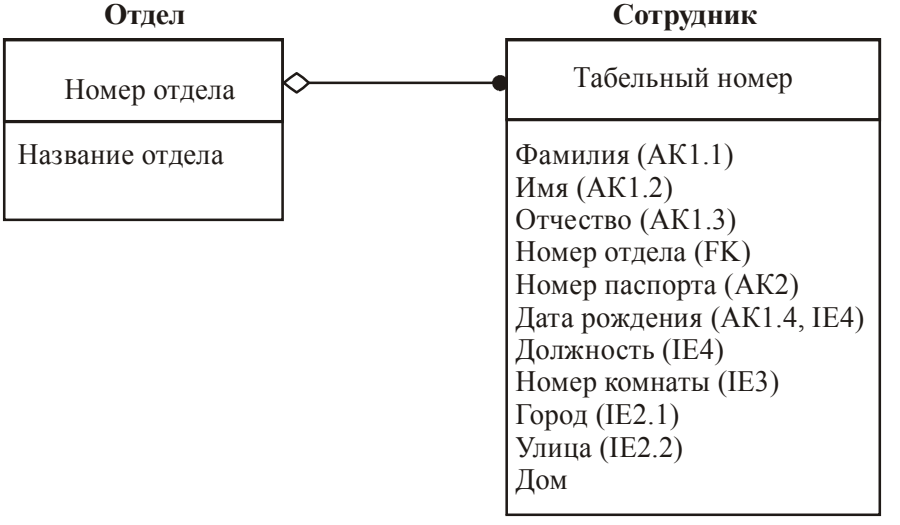


Рисунок 1.27 – Сущность «Сотрудник» с отображением ключей

Если один атрибут входит в состав нескольких ключей, ключи перечисляются в скобках через запятую (атрибут *Дата рождения* входит в состав IE4 и АК1). По умолчанию номера альтернативных ключей и инверсионнных входов рядом с именем атрибута на диаграмме не показываются.

Внешние ключи (Foreign Key) создают автоматически, когда связь соединяет сущности. Связь образует ссылку на атрибуты первичного ключа в дочерней сущности, и эти атрибуты образуют внешний ключ в дочерней сущности (миграция ключа). Атрибут внешнего ключа обозначается символом (FK) после своего имени (см. рисунок 1.27). Атрибут внешнего ключа *Номер отдела* является атрибутом первичного ключа (PK) сущности *Отдел*.

Рассмотрим пример описания предметной области для проектирования базы данных: государственная автоинспекция города Самары ведет учет транспортных средств, учет владельцев транспортных средств и нарушений, которые совершаются владельцами транспортных средств, находящимися за рулем своей автомашины. Транспортное средство характеризуется государственным номером, мощностью двигателя, маркой, датой выпуска. Владелец транспортного средства характеризуется номером паспорта, ФИО владельца, адресом с указанием района проживания, телефонами (сотовым, рабочим и/или домашним). Нарушение характеризуется номером протокола, местом на-

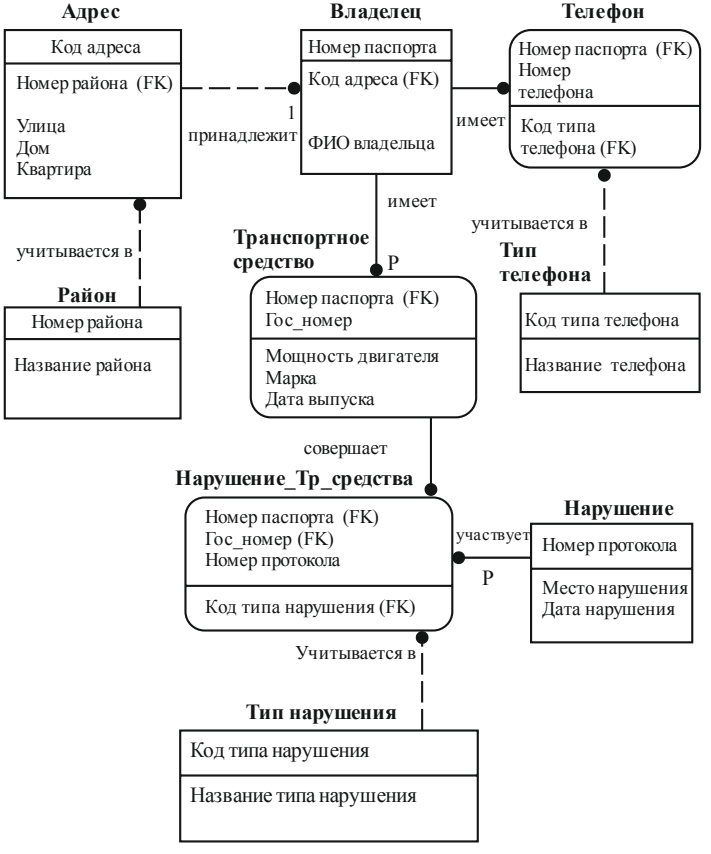


Рисунок 1.28 – Полная атрибутивная логическая модель базы данных «ГИБДД»

рушения, типом нарушения, датой. На рисунке 1.28 описана полная атрибутивная логическая модель базы данных в терминах методологии IDEF1X. Для обеспечения целостности данных введены сущности «Тип нарушения», «Район», «Тип телефона» которые являются справочниками и позволяют сократить возможные ошибки при наборе информации введением возможности выбора из справочников нужных значений, а также использованием одного и того же значения справочника многократно для разных сущностей.

Кроме нотации IDEF1X распространенной в настоящее время является и нотация IE. На рисунке 1.29 приведено соответствие графических примитивов связей в обеих нотациях.

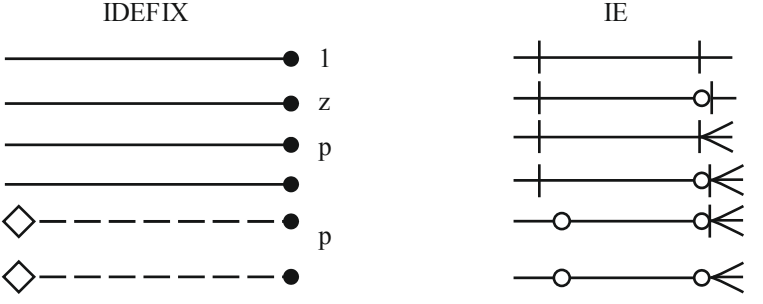


Рисунок 1.29 – Соответствие графических примитивов связей нотаций IDEF1X и IE для моделей «сущность-связь»

1.5. CASE-средства проектирования баз данных

Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности систем баз данных. Опыт проектирования таких систем показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Начиная с 70-х и 80-х годов, при разработке информационных систем широко применяется структурная методология, предоставляющая в распоряжение разработчиков строгие формализованные методы описания систем и принимаемых технических решений. Она основана на наглядной гра-

фической технике: для описания различного рода моделей используются схемы и диаграммы. Для автоматизации этой технологии в настоящее время используются программно-технологические средства специального класса – CASE-средства, реализующие CASE-технологию создания и сопровождения информационных систем. Термин CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь программного обеспечения, в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных автоматизированных систем в целом.

CASE-средства – это автоматизированные средства, основанные на CASE-технологиях, позволяющие автоматизировать отдельные этапы жизненного цикла программного обеспечения. Все современные CASE-средства могут быть классифицированы по типам и категориям. Классификация по типам отражает функциональную ориентацию на процессы жизненного цикла программного обеспечения. Классификация по категориям определяет степень интеграции по выполняемым функциям и включает отдельные локальные средства, решающие наиболее автономные задачи (по-английски tools), набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла (toolkit), и полностью интегрированные средства, поддерживающие весь жизненный цикл информационных систем.

Классификация по типам включает следующие основные CASE-средства:

1) средства анализа, предназначенные для построения и анализа моделей предметной области (Bpwin, Design/IDEF);

2) средства анализа и проектирования, предназначенные для создания проектных спецификаций (CASE.Аналитик, Vantage Team Builder, Designer/2000, Silverrun, PRO-IV);

3) средства проектирования баз данных, обеспечивающие моделирование данных и генерацию схем баз данных для наиболее распространенных СУБД (Silverrun, Vantage Team Builder, Designer/2000, ERwin, S-Designor);

4) средства разработки приложений и генераторы кодов (Vantage Team Builder, Silverrun, PRO-IV);

5) средства реинжиниринга, обеспечивающие анализ программных кодов, схем баз данных и формирование на их основе различных моделей и проектных спецификаций. Средства анализа схем баз данных входят в состав: (Silverrun, Vantage Team Builder, Designer/2000, Erwin, S-Designor). Для анализа программных кодов используются такие средства, как Rational Rose и Object Team.

В контексте данного учебного пособия наиболее интересны CASE-средства, используемые при проектировании баз данных, перечисленные в пункте 3.

CASE-средство Silverrun американской фирмы Computer Systems Advisers (CSA) используется для анализа и проектирования информационных систем бизнес-класса и ориентировано, в большей степени, на спиральную модель жизненного цикла. Оно применимо для поддержки любой методологии, основанной на раздельном построении функциональной и информационной моделей (диаграмм потоков данных и диаграмм «сущность-связь»). Silverrun имеет модульную структуру и состоит из четырех модулей, каждый из которых является самостоятельным продуктом. Модуль построения моделей бизнес-процессов в форме диаграмм потоков данных (BMP – Business Process Modeler) позволяет моделировать функционирование обследуемой организации или создаваемой информационной системы. Модуль концептуального моделирования данных (ERX – Entity-Relationship eXpert) обеспечивает построение моделей данных «сущность-связь», не привязанных к конкретной реализации. Модуль реляционного моделирования (RDM – Relational Data Modeler) позволяет создавать детализированные модели «сущность-связь», предназначенные для реализации в реляционной базе данных. Менеджер репозитория рабочей группы (WRM – Workgroup Repository Manager) применяется как словарь данных для хранения общей для всех моделей информации, а также обеспечивает интеграцию модулей Silverrun в единую среду проектирования. Для автоматической генерации схем баз данных у Silverrun существуют мосты к наиболее распространенным СУБД: Oracle, Informix, DB2, Ingres, Progress, SQL Server, SQLBase, Sybase. Для

передачи данных в средства разработки приложений имеются мосты к языкам 4GL: JAM, PowerBuilder, SQL Windows, Uniface, NewEra, Delphi. Система Silverrun реализована на трех платформах – MS Windows, Macintosh, OS/2 Presentation Manager – с возможностью обмена проектными данными между ними.

Vantage Team Builder представляет собой интегрированный программный продукт, ориентированный на реализацию каскадной модели жизненного цикла программного обеспечения. Vantage Team Builder обеспечивает выполнение следующих функций: 1) проектирование диаграмм потоков данных, «сущность-связь», структур данных, структурных схем программ и последовательностей экранных форм; 2) генерацию кода программ на языке 4GL целевой СУБД с полным обеспечением программной среды и генерация SQL-кода для создания таблиц баз данных, индексов, ограничений целостности и хранимых процедур; 3) программирование на языке C со встроенным SQL; 4) управление версиями и конфигурацией проекта; 5) генерация проектной документации по стандартным и индивидуальным шаблонам; 6) экспорт и импорт данных проекта. Vantage Team Builder поставляется в различных конфигурациях в зависимости от используемых СУБД (Oracle, Informix, Sybase, Ingress) или средств разработки приложений (Uniface).

CASE-средство Designer/2000 фирмы Oracle является интегрированным CASE-средством, обеспечивающим в совокупности со средствами разработки приложений Developer/2000, поддержку полного жизненного цикла программного обеспечения для систем, использующих СУБД Oracle. В состав Designer/2000 входят следующие компоненты: 1) Repository Administrator – средства управления репозиторием (создание, удаление приложений, управление доступом к данным со стороны различных пользователей, экспорт и импорт данных); 2) Repository Object Navigator – средства доступа к репозиторию, обеспечивающие многооконный объектно-ориентированный интерфейс доступа ко всем элементам репозитория; 3) Process Modeller – средство анализа и моделирования деловой деятельности, основывающиеся на концепциях реинжиниринга бизнес-процессов и глобальной системы управления качеством; 4) Systems Modeller – набор средств

построения функциональных и информационных моделей проектируемой информационной системы, включающий средства для построения диаграмм «сущность-связь», диаграмм функциональных иерархий, диаграмм потоков данных и средство анализа и модификации связей объектов репозитория различных типов; 5) Systems Designer – набор средств проектирования информационных систем, включающий средство построения структуры реляционной базы данных, а также средства построения диаграмм, отображающих взаимодействие с данными, иерархию, структуру и логику приложений, реализуемую хранимыми процедурами на языке SQL; 6) Server Generator – генератор описаний объектов базы данных Oracle (таблиц, индексов, ключей, последовательностей и т.д.). Помимо продуктов Oracle, генерация и реинжиниринг баз данных может выполняться для СУБД Informix, DB/2,Microsoft SQL Server, Sybase,а также для баз данных, доступ к которым реализуется посредством ODBC; 7) Forms Generator – генератор приложения, включающий в себя различные экранные формы, средства контроля данных, проверки ограничений целостности и автоматические подсказки; 8) Repository Reports – генератор стандартных отчетов. Среда функционирования Designer/2000 – Windows 3.x, Windows 95, Windows NT.

Erwin – средство логического моделирования баз данных, использующее методологию IDEF1X. Erwin реализует проектирование схемы баз данных, генерацию её описания на языке целевой СУБД (Oracle, Informix, DB/2, Ingres, Progress, SQL Server, SQLBase, Sybase и др.) и реинженириng существующей базы данных. Erwin выпускается в нескольких различных конфигурациях, ориентированных на наиболее распространенные средства разработки приложений 4GL. Версия Erwin/Open полностью совместима со средствами разработки приложений PowerBuilder и SQLWindows и позволяет экспортировать описание спроектированной базы данных непосредственно в репозитории данных средств.

S –Designor представляет собой CASE – средство для проектирования реляционных баз данных. S –Designor реализует стандартную методологию моделирования данных и генерирует описание баз данных для таких СУБД, как Oracle, Informix, DB/2, Ingres, Progress, SQL

Server, SQLBase, Sybase и др. Для существующих систем выполняется реинжениринг баз данных.

Из перечисленных средств универсальными средствами, ориентированными только на проектирование баз данных, являются последние два.

В следующем разделе будут рассмотрены теоретические аспекты реляционных баз данных, влияющие на выбор решений при разработке баз данных и последующей работе с базами данных.

Глава 2. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

2.1. Основные понятия. Операции обновления и реляционной алгебры

Основоположником теории реляционных баз данных является американский ученый Кодд. Ниже приведены основные понятия этой теории.

Домен – это множество однотипных значений данных. Например, домен целых чисел, домен имен студентов, домен символьных строк определенной длины. Несколько атрибутов одного отношения могут получать значение из одного домена. При этом с точки зрения логики значения домена по-разному интерпретируются в разных атрибутах. Например, доменом целых чисел можно представить атрибуты «оценка» и «год рождения» сущности «Студент».

Отношение R – это множество упорядоченных наборов данных (кортежей), являющихся подмножеством декартового произведения доменов (домены необязательно различные). Кортежи обозначаются перечислением значений доменов, заключенных в угловые скобки. Домены обозначаются заглавной латинской буквой D. Отсюда отношение $R \subseteq D1 \times D2 \times D3 \times \ldots \times Dn$. Почему в определении отношения говорится о подмножестве доменов? Рассмотрим отношение «Студент» (рисунок 2.1), включающее два атрибута «Имя» и «Возраст».

ИМЯ (домен символьных строк D1 длиной меньше 15)	ВОЗРАСТ (домен целых чисел D2)
Галя	17
Ира	18
Света	19

Рисунок 2.1 – Отношение «Студент»

Операция декартового произведения кортежей включает множество кортежей вида: D1 × D2 = <Галя, 17>, <Галя, 18>, <Галя, 19> … <Света, 19> . На самом деле имени соответствует один возраст, то есть $R \subseteq D1 \times D2$. С другой стороны, не всё множество значений домена используется в отношении, что также подтверждает определение отношения как подмножества декартового произведения домены.

Схема отношения – это перечень имён атрибутов отношения. Обозначается схема как *R (A1, A2, ..., An)*, где A_i – номера или имена атрибутов. Для примера отношения «Студент» схема имеет вид R(Имя, Возраст) или R(A1,A2), где A1=Имя, A2=Возраст.

Мощность отношения – число кортежей отношения. Для отношения «Студент» мощность равна трем.

Степень отношения – число атрибутов отношения. Для отношения «Студент» степень равна двум.

Каждое отношение должно удовлетворять следующим требованиям:

1) отношение имеет фиксированное число и расположение атрибутов;
2) отношение имеет конечное множество кортежей;
3) в отношении не может быть одинаковых кортежей.

Всё множество операций над отношениями можно разделить на следующие группы: операции обновления, операции реляционной алгебры, реляционное исчисление с переменными-кортежами, реляционное исчисление с переменными-доменами.

К *операциям обновления* относятся операции добавления, удаления, изменения кортежей данного отношения.

Пусть имеется отношение R(A1,A2,…,An). Операция добавления кортежа в это отношение выглядит следующим образом:

ADD(R; A1=d1, A2=d2, …,An=dn) или

ADD(R; d1,d2,…,dn),

где di, i=1,2,…,n – компоненты добавляемого кортежа <d1,d2,…,dn>.

При выполнении операции добавления возможно возникновение следующих ошибочных ситуаций:

1) добавляемый кортеж не соответствует схеме указанного отношения;

2) некоторые значения кортежа не принадлежат соответствующим доменам;
3) добавляемый кортеж совпадает по ключу с кортежем, уже находящимся в отношении.

В каждом из перечисленных случаев операция добавления оставляет отношение без изменения.

Операция удаления кортежа из отношения записывается в виде

DEL(R; A1=d1, A2=d2, …,An=dn) или

DEL(R; d1,d2,…dn),

где di, i=1,2,…,n – компоненты удаляемого кортежа <d1,d2,…,dn> .

Для удаления кортежа можно указать только значения ключевых атрибутов:

DEL(R; B1=l1, B2=l2, …,Bn=ln) или DEL(R; l1,l2,…,ln),

где $\{B_1, B_2, ..., B_k\}$ – перечень ключевых атрибутов,

lj $\in \{d_1, d_2, ...d_n\}$-значения ключевых атрибутов (j=1,2,…,k).

Операция удаления всегда выполняется успешно. В случае, если удаляемого кортежа нет, то отношение остаётся без изменений. Операция изменения в некотором кортеже отношения одних атрибутов на другие записывается в виде

CH(R; A1=d1, A2=d2, …,An=dn; C1=e1,C2=e2,…,Cp=ep).

Операция изменения может быть выполнена с помощью последовательности операций удаления и добавления, поэтому ошибочные ситуации этих операций присущи и операции изменения.

Рассмотренные выше операции выполнялись над отдельными кортежами и в результате давали отношение с той же самой схемой. Следующие операции реляционной алгебры выполняются целиком над отношениями и в результате порождают отношения с новой схемой. Так как отношение является подмножеством, то все операции над множествами выполняются и над отношениями. Для некоторых операций необходимо использование совместимых отношений. Два отношения называются *совместимыми отношениями*, если число и состав их атрибутов совпадают, то есть схемы отношений одинаковые.

1. *Операция объединения отношений R=R1 ∪ R2*, выполняется над совместимыми отношениями. В результирующее отношение входят кортежи первого и второго отношений. Повторяющиеся кортежи не включаются. На рисунке 2.2 показан пример выполнения операции объединения.

R1: Группа 634 (прикладная математика)	
Фамилия	Возраст
Раудин	19
Кайрис	18
Осин	19

R2: Группа 633 (физика)	
Фамилия	Возраст
Гашников	19
Дюмина	18
Москаленко	19
Осин	19

R1 R2

R= R1 ∪ R2

Фамилия	Возраст
Гашников	19
Дюмина	18
Москаленко	19
Осин	19
Раудин	19
Кайрис	18

Рисунок 2.2 – Операция объединения отношений

2. *Операция пересечения над отношениями R= R1 ∩ R2* также выполняется над совместимыми отношениями. В итоговое отношение включаются кортежи, принадлежащие как первому, так и второму отношениям. На рисунке 2.3 показан результат выполнения операции пересечения.

.

Фамилия	Возраст
Осин	19

Рисунок 2.3 – Результат выполнения операции пересечения
R= R1 ∩ R2

3. *Операция разности над отношениями R=R1\R2* (или *R1-R2*), выполняется над совместимыми отношениями. В результирующее отношение включаются только те кортежи первого отношения, которых нет во втором отношении. На рисунке 2.4 показан результат операции разности отношений

Фио	Возраст
Раудин	19
Кайрис	18

Рисунок 2.4 – Результат выполнения операции разности R=R1-R2

4. *Операция декартового произведения R = R1 × R2*. Здесь R1 и R2 могут иметь разные схемы. Степень отношения R равна сумме степеней отношений R1 и R2. Мощность отношения R равна произведению мощностей отношений R1 и R2. На рисунке 2.5 показан пример выполнения операции декартового произведения над отношениями.

5. *Операция деление над отношениями R = R1:R2*. Отношение-делимое R1 должно содержать в качестве подмножества множество атрибутов отношения делителя. Частное R содержит только те атрибуты делимого, которых нет в делителе. В отношение-частное включены только те кортежи, декартовое произведение которых с отношением-делителем содержатся в отношении-делимом.

$$R = \begin{bmatrix} x\,a \\ y\,a \\ z\,a \end{bmatrix} \quad F = \begin{bmatrix} 1\,a\,1 \\ 2\,b\,1 \end{bmatrix} \quad R \times F = \begin{bmatrix} x & a & 1 & a & 1 \\ x & a & 2 & b & 1 \\ y & a & 1 & a & 1 \\ y & a & 2 & b & 1 \\ z & a & 1 & a & 1 \\ z & a & 2 & b & 1 \end{bmatrix}$$

Рисунок 2.5 – Пример выполнения операции декартового произведения над отношениями

На рисунке 2.6 показан пример выполнения операции деления над отношениями.

R1

Фамилия	Пред-мет	Оценка
Иванов	Оит	5
Иванов	Физика	5
Петров	Оит	4
Петров	Физика	4
Сидоров	Оит	5
Сидоров	Физика	4

R2

Предмет	Оценка
Оит	5
Физика	4

Фамилия
Сидоров

R=R1:R2

Рисунок 2.6 – Пример выполнения операции деления над отношениями

Следующие операции характерны только для отношений.

6. *Операция проекции* на некоторые атрибуты – это операция выборки из каждого кортежа отношения R значений атрибутов, входящих в некоторое множество атрибутов – A и удаление из полученного отношения повторяющихся строк. Операция проекции унарная, выполняется над одним отношением:

$$R = \prod i_1, i_2, ..., i_r (R_1),$$

где $i_1, ..., i_r$ – номера или имена атрибутов отношения R_1, входящих в результирующее отношение. На рисунке 2.7 приведен пример выполнения операции проекции над отношением R, со схемой R(фамилия, предмет, оценка).

Проекция $R = \prod предмет, оценка(R1) = \prod 2,3(R1)$

Фамилия	Предмет	Оценка
Иванов	Оит	5
Иванов	Физика	5
Петров	Оит	5
Петров	Физика	4

Предмет	Оценка
Оит	5
Физика	4
Физика	4

Рисунок 2.7 – Пример выполнения операции проекции над отношением R

7. *Операция соединение* является двухместной операцией, то есть выполняется над двумя отношениями. В каждом отношении выделяется атрибут, по которому будет производиться соединение. Эта операция отличается от декартового произведения тем, что комбинируются лишь те кортежи отношений, у которых значения совпадающих атрибутов равны. Обозначается эта операция следующим образом:R=R1►◄R2. На рисунке 2.8 приведен пример выполнения операции соединения отношений R1 и R2 по атрибуту «Код студента».

Приведенный вид операции соединения называют *операцией внутреннего соединения*. В результирующее отношение вошли кортежи с совпадающими значениями поля соединения. Для операции соединения существенен порядок записи отношений в операции. Кроме операции внутреннего соединения различают операции левого внешнего соединения, правого внешнего соединения и полного соединения отношений. В результате выполнения *операции левого внешнего соединения* получается отношение, включающее все кортежи левого отношения, при этом кортежи с неизвестными значениями поля соединения заполняются значениями Null (рисунок 2.9).

R1

Специальность	Код студента
Математика	1
Физика	2
Оит	3

R2

Код студента	Фамилия	Курс
1	Иванов	1
2	Петров	2
3	Сидоров	3

R=R1►◄R2

Специальность	Код студента	Фамилия	Курс
Математика	1	Иванов	1
Физика	2	Петров	1

Рисунок 2.8 – Пример выполнения операции внутреннего соединения отношений

В результате выполнения *операции правого внешнего соединения* получается отношение, включающее все кортежи правого отношения,

при этом кортежи с неизвестными значениями поля соединения заполняются значениями Null (рисунок 2.10).

Специальность	Код студента	Фамилия	Курс
Математика	1	Иванов	1
Физика	2	Петров	1
Оит	3	Null	Null

Рисунок 2.9 – Пример выполнения операции
левого внешнего соединения отношений

Специальность	Код студента	Фамилия	Курс
Математика	1	Иванов	1
Физика	2	Петров	1
Null	4	Сидоров	3

Рисунок 2.10 – Пример выполнения операции
правого внешнего соединения отношений

В результате выполнения *операции полного соединения* двух отношений получают отношение, включающее кортежи обоих отношений. Причем в одном экземпляре входят кортежи с совпадающим значением поля соединения и кортежи, для которых значение поля соединения не совпадает, включаются из обоих отношений, причем для них неизвестные значения полей заполняются значениями Null. На рисунке 2.11 показан пример выполнения операции полного соединения.

Специальность	Код студента	Фамилия	Курс
Математика	1	Иванов	1
Физика	2	Петров	1
Оит	3	Null	Null
Null	4	Сидоров	3

Рисунок 2.11 – Пример выполнения операции
полного соединения отношений

Существует еще разновидность операции соединения, называемая операцией *θ-соединения*, в которой в отличие от обычной операции соединения для соединения используется не один атрибут, а несколько атрибутов, принадлежащих обоим отношениям. В свою очередь разновидностью θ – соединения является операция эквисоединения. При соединении могут использоваться различные операции сравнения над подмножеством атрибутов, а не только равенство, как в эквисоединении. Следует отметить, что при выполнении операций соединения наименования атрибутов, по которым выполняется соединение, могут не совпадать, зато домены, с помощью которых задаются атрибуты связи, должны быть идентичными.

8. *Операция селекции* является операцией над несколькими отношениями. Результирующее отношение содержит подмножество кортежей, выбранных по некоторому условию:

$$R = \delta_F \left(R_1, ..., R_n \right),$$

где F – формула, включающая:

1) имена атрибутов и константы различных типов;
2) логические операции и кванторы всеобщности: $\wedge, \forall, \exists, \vee, \neg$;
3) арифметические операции и операции сравнения: $\leq, \geq, <, >, =, \neq$;
4) скобки.

Например, дано отношение R1:

Фамилия	Оценка	Предмет
Иванов	5	Оит
Иванов	5	Физика
Петров	5	Оит

R= $\delta_{(оценка>4)\wedge(предмет=оит)}(R_1)$ – отношение, включающее два кортежа: <Иванов,5,Оит>,<Петров, 5, Оит >.

Современные языки манипулирования данными в реляционных базах данных включают команду Select, название которой совпадает с

названием рассмотренной операции и, по сути, названная команда аналогично операции селекции осуществляет выборку данных из базы данных, удовлетворяющих некоторому предикату.

9. Операция перемены местами позиций $(i \leftrightarrow j)R$ создает новое отношение, в котором изменен порядок расположения атрибутов исходного отношения.

10. Операция расширения исходного отношения атрибутом. Последние две операции позволяют изменять структуру отношения.

Перечисленные операции относятся к операциям реляционной алгебры. Языки реляционной алгебры – процедурные языки программирования. Существуют языки реляционного исчисления, не являющиеся процедурными, основанные на классическом исчислении предикатов – эти языки дают набор правил для записи запросов к базе данных. В запросе – информация о желаемом результате. На основании запроса СУБД автоматически, путем формирования новых отношений, выдает желаемый результат. Это непроцедурные языки (в частности, к ним относится и язык запросов SQL).

2.2. Реляционное исчисление кортежей и доменов

В то время как реляционная алгебра использует в качестве операндов отношения, реляционное исчисление кортежей строит свои выражения над переменными, принимающими значения кортежей (переменные-кортежи). Выражения *реляционного исчисления кортежей* имеют вид

$$\{x(R) \mid f(x)\},$$

где f-формула, выражающая некоторый предикат над переменной-кортежем x . Результат этого выражения есть отношение r(R),которое состоит из всех кортежей t(R), для которых f(t) истинно.

Минимальные строительные блоки для формул называют атомами. Атомы бывают следующих типов:

1) атом $x \in R$ *или* $R(x)$, где x – кортеж отношения R;

2) атом *S(i) θ U(j), где S* и *U* – переменные-кортежи; *θ* – оператор сравнения; *i, j* – номера или имена атрибутов в соответствующих кортежах;
3) атом *S(i) θ a* или, *a θ S(i), где, a* – константа, *S* – переменная-кортеж; *θ* – оператор сравнения; *i* – номер или имя атрибута в соответствующем кортеже;
4) операции: сравнения ($\leq, \geq, <, >, =, \neq$), кванторы всеобщности ($\forall, \exists$), логические операции($\wedge, \vee, \neg$.);
5) скобки.

В реляционном исчислении с переменными-кортежами справедливо следующее утверждение: для любого выражения реляционной алгебры существует эквивалентное ему безопасное выражение в реляционном исчислении с переменными-кортежами. В исчислении кортежей на вид формул накладываются определенные ограничения, чтобы исключить выражения, не имеющие смысла. Формулы, удовлетворяющие этим ограничениям, называют разрешёнными формулами, а соответствующие выражения – безопасными.

Реляционное исчисление доменов напоминает реляционное исчисление кортежей, за тем исключением, что переменные этого исчисления являются переменными значениями из доменов, а не переменными-кортежами. Выражение исчисления доменов имеет вид

$$\left\{ x_1(A_1)x_2(A_2)\ldots x_n(A_n) \mid f\left(x_1,\ldots,x_n \right)_k \right\},$$

где *f* – формула, обладающая тем свойством, что ее переменные являются переменными значениями из доменов. Формула состоит из следующих атомов:

1) атом $R(a_1,\ldots,a_k)$, где a_i – либо переменная исчисления доменов, либо постоянная из домена, с помощью которого задается атрибут;
2) атом $\boldsymbol{x\ \theta\ y}$, где *x, y* – константы или переменные на домене; *θ* – оператор сравнения;

3) формула использует кванторы $\forall x; \exists x$ и логические операции $\wedge, \vee, \neg$;

4) скобки.

В реляционном исчислении с переменными на доменах справедливы следующие утверждения:

1. Для каждого безопасного выражения реляционного исчисления с переменными кортежами существует эквивалентное ему безопасное выражение реляционного исчисления с переменными на доменах.

2. Для каждого безопасного выражения реляционного исчисления с переменными на доменах существует эквивалентное ему выражение реляционной алгебры.

2.3. Языки манипулирования данными в реляционных системах

Рассмотренные выше реляционная алгебра и реляционные исчисления над кортежами и доменами явились теоретической основой для реализации языков манипулирования данными в реляционных системах. Далее приводятся примеры наиболее известных языков, используемых в реляционных СУБД.

1. На основе реляционной алгебры фирмой IBM был разработан язык ISBL(Information System Base Language). Выражения этого языка строятся с помощью операций, соответствующих операциям реляционной алгебры или их обобщениям. Например, объединение, пересечение, соединение и выбор обозначаются соответственно: $+(\cup); \bullet(\cap); *(\infty); \sigma_F(R)$ *(R : F)*.

Результат вычисления любого выражения языка ISBL может быть присвоен некоторому отношению, либо напечатан с помощью ключевого слова list.

2. Реляционное исчисление кортежей послужило основой для создания языка QUEL(QUEry Language), разработанного в университете шт. Калифорния. Все переменные языка неявно связаны квантором существования, и область определения каждой из них ограниче-

на одним из отношений. Объявление переменной и её привязка к определенному отношению производится с помощью оператора следующего вида:

range of <переменная> is <имя отношения>

Основной формой оператора запроса в QUEL является:

retrieve (<целевой список>) where <условие>,

где условию соответствует формула исчисления кортежей. Целевой список представляет собой последовательность компонент переменных, напоминающую ту часть выражения в исчислении доменов, что расположена слева от вертикальной черты. Результат запроса можно присвоить некоторому отношению, если после ключевого слова retrieve написать into <отношение>.

3. Слабость языка QUEL заключается в том, что он использует для исчисления кортежей только кванторы существования. Это послужило причиной роста популярности другого языка, основанного на полном исчислении кортежей и частично реляционной алгебре – язык SQL. Язык SQL был разработан в научно-исследовательской лаборатории фирмы IBM в Сан-Хосе (шт. Калифорния). Основной операцией в SQL служит отображение, представляющее собой композицию ограничения предикатом и проекции. В простейшем случае отображение выражается синтаксической конструкцией вида

select <список атрибутов>
from <отношение>
where <условие>.

4. Реляционное исчисление доменов стало основой для создания языка QBE(Query-By-Example) – реляционного языка манипулирования данными, разработанного Злуфом из Уотсоновского исследовательского центра фирмы IBM. Язык QBE обладает двумерным синтаксисом. Запрос формулируется путём заполнения табличной формы,

содержащей имя отношения и имена атрибутов схемы. Строки запроса имеют вид

Имя отношения	Имя атрибута 1	….	Имя атрибута n
операция			

Примером СУБД, реализующей язык QBE, является Paradox.

Каждый из рассмотренных языков обладает своими особенностями. Языки, основанные на реляционных исчислениях, – непроцедурные языки, поскольку их средствами можно описать то, что необходимо и не обязательно указывать, как это получить. Выражения реляционной алгебры задают конкретный порядок выполнения операций. В общем случае языки манипулирования данными реальных СУБД выходят за рамки абстрактных языков, так как для обработки данных требуются операции, выходящие за рамки возможностей реляционной алгебры или реляционного исчисления. К таким командам относятся команды добавления, изменения, удаления, а также команды присваивания, печати, функции и другие.

2.4. Понятие ключа и функциональных зависимостей

Функциональные зависимости отображают смысловые, семантические связи между атрибутами отношения и в теории проектирования реляционных баз данных играют важную роль. Существует множество видов функциональных зависимостей между атрибутами отношений.

Атрибут B функционально зависит от атрибута A ($A \rightarrow B$*)*, если для каждого значения атрибута *A* существует ровно одно, связанное с ним значение атрибута *B*. *A* и *B* могут быть составными атрибутами. Атрибут называется составным, если он включает множество атрибутов отношения.

Если $A = A_1A_2...A_к$ и $A \rightarrow B$, то есть B функционально зависит от всей группы атрибутов A, то говорят, что *B функционально полно зависит от A*.

Если $A = A_1A_2...A_к$ и $A_i...A_j \rightarrow$ B и $i, j < k$, то *между A и B имеется частичная функциональная зависимость*.

Если $A \leftrightarrow B$, то *A и B взаимозависимые атрибуты*.

Взаимозависимость также может быть полной и частичной.

Если для атрибутов *A, B и C* выполняется условие $A \rightarrow$ B и $B \rightarrow C$, но обратной связи нет, то говорят, что *C транзитно зависит от A*.

Атрибут B многозначно зависит от A, $A \rightarrow\rightarrow B$, если каждому значению *A* соответствует множество значений *B*, никак не связанных с другими атрибутами отношения.

Ключом отношения называется непустое подмножество атрибутов, от которого функционально полно зависит все множество атрибутов отношения. Аналогом введенного понятия является понятие потенциального ключа (см. раздел 1.3). В отношении может быть множество потенциальных ключей, среди которых выбирается один первичный. Впоследствии, называя ключ, будем понимать потенциальный ключ отношения.

2.5. Нормализация отношений. 1, 2, 3, 4, 5 нормальные формы отношений

Нормализация – это процесс разбиения или декомпозиции исходного отношения на несколько отношений с целью устранения нежелательных функциональных зависимостей, приводящих к возникновению избыточности хранения информации и аномалиям добавления, удаления, обновления.

Аппарат нормализации отношений разработан Коддом. В нем определены различные *нормальные формы отношений*. Каждая нормальная форма ограничивает типы допустимых функциональных зависимостей отношений. Кодд выделил три нормальные формы: 1, 2, 3НФ, а сегодня определены 4НФ, 5НФ.

Следует отметить, что нормализация или иначе декомпозиция должна быть обратимой, то есть выполняться без потерь информации. Вопрос о том, происходит ли утрата информации при декомпозиции, тесно связан с концепцией функциональной зависимости. В качестве примера рассмотрим отношение поставщик (номер поставщика, статус поставщика, город проживания поставщика). В отношении 2 кортежа:(«S1, 30, Париж», «S2, 30, Лондон»). Выполним 2 варианта декомпозиции исходного отношения: а) R1(номер, статус), R2(номер, город); б) R1(номер, статус), R2(статус, город). Первая декомпозиция получилась без потерь, а другая с потерей информации. Операция декомпозиции выполнялась путём применения дважды операции проекции к исходному отношению. В случае а) при обратном соединении полученных отношений получается исходное отношение. В случае б) при соединении полученных отношений исходное отношение не получается, а значит некоторая информация утрачена. Это произошло из-за потери одной функциональной зависимости (ФЗ), присутствующей в исходном отношении. Действительно, в исходном отношении имеются следующие ФЗ: номер → статус и номер → город. В случае а) обе функциональные зависимости сохраняются, в б) вторая ФЗ потеряна, в результате чего при выполнении операции соединения R1 и R2 произойдет потеря информации, так как неизвестно, в каком городе проживают поставщики с одинаковыми статусами, но разными номерами. Зависимость между декомпозицией без потерь ФЗ подтверждается теоремой Хеза.

Теорема Хеза: Пусть R(A,B,C) является отношением с атрибутами A,B,C. Если оно удовлетворяет зависимости A → B, то R равно соединению его проекций (A,B) и (A,C).

1НФ – отношение находится *в первой нормальной форме*, если значения всех атрибутов атомарные, то есть значение атрибута не является множеством, диапазоном или повторяющейся группой.

2НФ – отношение находится *во второй нормальной форме*, если оно находится в первой нормальной форме и отсутствуют частичные функциональные зависимости атрибутов, не входящих в ключ от ключа (зависимости атрибутов, не входящих в ключ от части ключа). Иначе можно сказать, если каждый атрибут, не входящий в ключ, функ-

ционально полно зависит от ключа. Анализируя определение второй нормальной формы, можно сделать вывод, что отношение находится во второй нормальной форме, если в нем отсутствуют составные ключи.

Рассмотрим на примере обоснование того, почему наличие этих частичных функциональных зависимостей нежелательно при работе с отношением. Пусть имеется отношение Поставки, в котором имеется информация о поставщике, товаре и его цене. Каждый товар имеет одну цену и может поставляться несколькими поставщиками.

Поставщик	Товар	Цена
Иванов	Мыло	2000
Иванов	Сахар	2500
Петров	Мыло	2000
Петров	Сыр	20000

В отношении имеется один составной ключ: <Поставщик, товар>. По определению ключа <Поставщик, товар> → цена. Кроме этого, имеется зависимость атрибута «цена» от части ключа: товар → цена, значит отношение не находится во второй нормальной форме. При наличии такой функциональной зависимости возникают следующие нежелательные случаи во время работы с отношением или иначе *аномалии включения, удаления, обновления*:

1. *Аномалия включения*: появился новый товар для поставки, но информация о товаре и его цене не сможет храниться в отношении, пока кто-нибудь не начнет поставлять его.

2. *Аномалия удаления*: прекратились поставки товара, тогда из базы надо удалить сведения о товаре и его цене, даже если этот товар имеется в наличии у поставщиков.

3. *Аномалия обновления*: изменилась цена товара, надо полностью посмотреть всё отношение, чтобы найти все поставки товара, чтобы изменение цены было отражено для всех поставщиков. То есть изменение значения одного объекта приводит к необходимости изменения в нескольких кортежах, иначе база будет несогласованной.

Кроме этого, в отношении имеется *избыточность хранения информации:* сведения о цене товара повторяются несколько раз для разных поставщиков, хотя по условию цена товара фиксирована. Причина

аномалий и избыточности хранения связана с наличием неполной функциональной зависимости атрибута цена от ключа, что обусловлено объединением в отношении поставки двух семантических фактов: сведения о товаре и поставщике объединены в одно отношение. Разложив отношение на два, выполнив операцию проекции над атрибутами, входящими в частичную функциональную зависимость, и удалив из исходного отношения неключевые атрибуты, зависящие от части ключа, устраним нежелательную функциональную зависимость, в результате чего получим два отношения.

Правило нормализации при приведении отношения ко второй нормальной форме: выполняется дважды операция проекции над исходным отношением. В результате первой операции получается отношение, в которое входит часть ключа и *все* атрибуты, от нее зависящие. В результате второй операции проекции получается отношение, в которое входит составной ключ и все оставшиеся атрибуты исходного отношения, не вошедшие в отношение, полученное после применения первой операции проекции.

Для рассмотренного примера в результате нормализации получим отношения R_1 (поставщик, товар) и R_2 (товар, цена). Отношение R1 имеет кортежи:<Иванов, мыло >, <Иванов, сахар>, <Петров, мыло >, <Петров, сыр>; отношение R2 имеет кортежи: <мыло,2000>, <сахар, 2500>, <сыр, 20000>. Анализируя возможность работы с базой данных «Поставки», при наличии двух отношений, видно, что можно добавить сведения о товаре и его цене до того, как кто-то начинает поставлять этот товар, сведения о товаре и его цене хранятся в базе один раз. То есть избыточность хранения устранена, аномалии устранены, кроме этого декомпозиция выполнена без потери информации.

3НФ – отношение находится *в третьей нормальной форме*, если оно находится во второй нормальной форме, и отсутствуют транзитивные зависимости атрибутов, не входящих в ключ от ключа. Иными словами, если выполняется совокупность условий:

$$A \to B;\ B \to C;$$
$$C \to \not{A};\ B \to \not{A};\ C \to \not{B},$$

тогда в отношении существует транзитивная зависимость атрибутов, не входящих в ключ от ключа A, и отношение не находится в третьей нормальной форме. Если хотя бы одно из условий не выполняется, то такой зависимости между атрибутами A,B,C нет. Причем атрибуты A,B,C могут быть составными.

Рассмотрим пример отношения «Хранение» с информацией о наименовании фирмы, о наименовании складов, принадлежащих фирмам, и объеме продукции на складах, причем каждая фирма получает товары с одного склада, склад определяется объемом. Таблица «Хранение» выглядит следующим образом:

Фирма	Склад	Объем
МЕНАТЕП	МКЧ	200
ИКС	ДП	600
АСКО	ДП	600
ПАРУС	СК	200

В данном отношении имеется один атомарный ключ <фирма>. Выполняются следующие функциональные зависимости: <фирма> $\rightarrow$ <склад>;<склад> $\rightarrow$ <объем>, <объем> $\not\rightarrow$ <фирма>;<склад> $\not\rightarrow$ <фирма>; <объем> $\not\rightarrow$ <склад>.

По определению, это отношение не находится в 3 НФ, в результате чего возникают избыточность хранения информации (если несколько фирм хранят товар на складе, то в данном отношении несколько раз повторяется информация об объеме склада) и аномалии:

Аномалия включения. Если надо ввести объем склада, а нет фирмы, получающей товар со склада, то добавление невозможно.

Аномалия удаления. Если фирма со склада не получает товар, тогда данные о складе и его объеме в базе хранить нельзя, то есть вместе с удалением сведений о фирме, хранящей товар на складе, будет удалена информация и о складе, если никакая другая фирма на этом складе не хранит товар.

Аномалия обновления. Если объем склада изменился, то надо просмотреть всё отношение и изменить все кортежи для фирм, связанных с этим складом, а это неэффективно.

Таким образом**,** наличие в отношении двух различных семантических фактов – сведений о фирме и складе – приводит к нежелательным функциональным зависимостям.

Правило нормализации отношения для приведения к третьей нормальной форме: дважды выполняется операция проекции над исходным отношением. В результате выполнения первой операции проекции в отношение включаются атрибуты B и C, не являющиеся ключами. В результате выполнения второй операции проекции над исходным отношением в отношение включаются ключевой атрибут A и все оставшиеся атрибуты исходного отношения, не попавшие в отношение, полученное в результате выполнения первой операции проекции.

Для рассмотренного примера выполнение правила приведения к третьей нормальной форме приведет к замене исходного отношения на два следующих отношения: R_1(фирма, склад); R_2(склад, объем). Такое разбиение является разбиением без потери информации. Если бы мы выполнили декомпозицию таким образом: R_1(фирма, склад); R_2(фирма, объем), то при соединении этих отношений было бы непонятно, к какому складу относится фирма, если склад имеет одинаковый объем, то есть происходит потеря информации.

4НФ – отношение находится в четвертой нормальной форме, если оно находится в третьей нормальной форме и в нем отсутствуют независимые многозначные функциональные зависимости.

Рассмотрим пример отношения «Преподаватель»:

Номер преподавателя	Дети преподавателя	Шифр курса
1	Ира	Математика
1	Вася	Математика
1	Коля	Физика
2	Петя	Математика

В отношении имеются один составной ключ <номер преподавателя, дети преподавателя, шифр курса> и многозначные функциональные зависимости:

<Номер преподавателя> $\rightarrow\!\!\rightarrow$ дети, <Номер преподавателя> $\rightarrow\!\!\rightarrow$ шифр курса. Причем многозначные функциональные зависимости не связаны между собой, так как дети $\not\rightarrow\!\!\rightarrow$ шифр курса, то есть отношение не находится в четвертой нормальной форме.

В результате нормализации получим: R1(номер преподавателя, дети); R2(номер преподавателя, шифр курса).

В отношениях со многими многозначными зависимостями 4НФ может не устранять избыточности, а вместе с тем и аномалий обновления. Тогда применяется 5НФ.

5НФ – отношение находится *в пятой нормальной форме*, если после декомпозиции отношения и его приведения к 4НФ оно удовлетворяет зависимости по соединению. Декомпозиция должна гарантировать обратимость, то есть обеспечивать получение исходных отношений путем выполнения операций соединения над их проекциями.

Обратимость предполагает, что отсутствуют потери кортежей, не появляются ранее отсутствующие кортежи, сохраняются функциональные зависимости. Не всегда декомпозиции гарантируют обратимость. При наличии в отношении более трех многозначных зависимостей требуются специальные меры для обеспечения зависимости по соединению его проекций. Для обеспечения 5НФ из 4НФ получают такие проекции, чтобы каждая проекция содержала не менее одного возможного ключа и, по крайней мере, один атрибут исходного отношения, не входящий в ключ.

Кроме описанных нормальных форм существуют и другие нормальные формы. В качестве примера ещё одной нормальной формы можно привести *доменно-ключевую нормальную форму – ДКНФ*, предложенную Фейгиным. Эта нормальная форма, в отличие от рассмотренных ранее, не определяется на основе функциональных зависимостей или многозначных зависимостей, или зависимостей соединения. Отношение находится в ДКНФ тогда и только тогда, когда каждое ограничение, наложенное на отношение, является логическим следствием ограничений доменом и ограничений ключей, наложенных на отношение. Под ограничением домена понимается ограничение на использование значений для данного атрибута из некоторого предписанного домена. Под ограничением ключа понимается ограничение на

то, что некоторый атрибут или их комбинация представляют собой первичный ключ. Наложение ограничений на отношение, находящееся в ДКНФ, является концептуально очень простым, поскольку для этого достаточно знать ограничения домена и ключа, а все остальные ограничения будут приведены в действие автоматически. Под выражением «все остальные ограничения» подразумевается нечто большее, чем просто функциональные и многозначные зависимости или зависимости соединения; это обозначает совокупность всех условий, накладываемых на отношение, выраженную в виде некоторого предиката. Фейгин доказал, что любое отношение, находящееся в ДКНФ, находится в 5НФ, но не всегда можно добиться приведения к ДНКФ или получить ответ на вопрос, когда такое приведение может быть выполнено.

Теория нормализации и связанных с ней вопросов, называемая также теорией зависимостей, развилась в значительную область знаний с несколькими различными направлениями. Другим направлением в исследованиях нормализации является применение декомпозиции на основе других операций, отличных от проекции. Например, для декомпозиции используется непересекающаяся выборка, а для композиции используется непересекающееся объединение. Примером реализации такого подхода является нормальная форма типа «выборка-объединение». Несмотря на то, что на практике структура отношений, полученная названным выше способом, получается хуже классической теории нормализации, строгого доказательства этому факту пока нет.

В следующем пункте рассмотрен формальный алгоритм приведения отношения к третьей нормальной форме на примере некоторого отношения.

2.6. Описание формального алгоритма приведения отношений к третьей нормальной форме

Рассмотрим формальный алгоритм приведения отношений к третьей нормальной форме на примере отношения, имеющего шесть атрибутов и восемь кортежей с определенными значениями. Будем

считать, что все особенности семантических функциональных зависимостей этого отношения учтены и отображаются в значениях полей атрибутов. Чтобы подчеркнуть, что имена атрибутов для формального алгоритма не существенны, будем называть их номерами и вынесем их в отдельную строку отношения. На рисунке 2.12 показано такое отношение R.

1	2	3	4	5	6
21	15	7	13	21	3
12	21	6	12	13	3
53	35	27	9	24	13
12	21	6	14	15	3
14	14	7	15	27	3
25	52	12	12	14	6
12	12	6	16	11	3
17	17	8	17	10	4

Рисунок 2.12 – Пример описания отношения R для демонстрации формального алгоритма приведения к третьей нормальной форме

Используя метод классификации, по каждому отдельному столбцу исходное отношение преобразуем к виду, описанному на рисунке 2.13. Метод классификации состоит в замене значений каждого поля столбца на меньшие значения, начиная с единицы. Это позволяет визуально проще выполнять анализ на уникальные значения или наличие функциональных зависимостей между атрибутами.

1	2	3	4	5	6
1	1	1	1	1	1
2	2	2	2	2	1
3	3	3	3	3	2
2	2	2	4	4	1
4	4	1	5	5	1
5	5	4	2	6	3
2	6	2	6	7	1
6	7	5	7	8	4

Рисунок 2.13 – Вид отношения R
после применения метода классификации

На первом шаге алгоритма определяем, находится ли отношение R в первой нормальной форме. Как видно, все значения атрибутов отношения имеют единственное значение, то есть не являются множеством или списком, отсюда следует, что отношение R находится в первой нормальной форме. Для анализа на вторую и третью нормальные формы необходимо определить все потенциальные ключи отношения (или просто ключи).

Для этого определяем по значениям отдельных атрибутов (столбцов) или совокупности атрибутов (столбцов) наличие таких атрибутов, у которых все значения не повторяются, то есть уникальные в пределах атрибута отношения. Как видно из рисунка 2.13, такими ключевыми атрибутами являются следующие атомарные и составные ключи: <5>,<1,4>,<2,4>,<3,4>,<4,6>.

Так как в отношении имеются составные ключи, то, возможно, это отношение не находится во второй нормальной форме. Начиная с первого составного ключа, проверяем наличие частичных функциональных зависимостей. У составного ключа <1,4> имеются две части – атрибуты 1 и 4. От части ключа могут зависеть атрибуты, не являющиеся ключами и не входящие с данным атрибутом в ключ для оставшихся составных ключей. Анализ наличия функциональной зависимости от атрибута 1 нужно выполнить для атрибутов 2,3,6. Определение функциональной зависимости дано в пункте 1.4. Как видно от атрибута 1 функционально зависят 3 и 6: 1→3, 1→6. Раз имеются зависимости от части ключа, то отношение не находится во второй нормальной форме. После приведения к 2НФ по правилу, получим два отношения R1(1,3,6) и R2(1,2,4,5). Оба этих отношения показаны на рисунке 2.14. Следует заметить, что повторяющиеся кортежи удалены, это необходимо сделать, так как по определению в отношении не допускаются одинаковые кортежи.

После этого шага для каждого из полученных отношений алгоритм повторяется, начиная с определения ключевых атрибутов. Сначала рассмотрим отношение R1. В этом отношении имеется один атомарный ключ <1>. Так как отсутствуют составные ключи, то отношение находится во второй нормальной форме. Выполним анализ на третью нормальную форму. В данном случае ключевой атрибут A –

это атрибут <1>. Для этого отношения выполняются соотношения: 1→3, 3→6, 6 -/->1, 6-/->3, 3-/->1.

R1(1,3,6)

1	3	6
1	1	1
2	2	1
3	3	2
4	1	1
5	4	3
6	5	4

R(1,2,4,5)

1	2	4	5
1	1	1	1
2	2	2	2
3	3	3	3
2	2	4	4
4	4	5	5
5	5	2	6
2	6	6	7
6	7	7	7

Рисунок 2.14 – После приведения
ко второй нормальной форме отношения R

Наличие перечисленных функциональных зависимостей показывает, что в отношении имеется транзитивная зависимость атрибутов, не являющихся ключом, от ключа и, значит, отношение не находится в третьей нормальной форме. Применение правила нормализации для третьей нормальной формы приводит к возникновению двух отношений R3 и R4 со следующими схемами R3(3,6), R4(1,3), показанными на рисунке 2.15.

R3(3,6)

3	6
1	1
2	1
3	2
4	3
5	4

R4(1,3)

1	3
1	1
2	2
3	3
4	4
5	5
2	6
6	7

Рисунок 2.15 – Схемы отношений после нормализации отношения R1

Полученные отношения R3 и R4 в третьей нормальной форме. Для анализа отношения на вторую нормальную форму в отношении должен быть как минимум один составной ключ из двух атрибутов и атрибут, не входящий в ключ, то есть не менее трех атрибутов. Для анализа на третью нормальную форму также в отношении должно быть не менее трех атрибутов, это следует из определения транзитивной функциональной зависимости.

Теперь проверим на наличие нормальных форм отношение R2(1,2,4,5). В этом отношении имеется следующий состав ключей: <5>,<1,4>,<2,4>. При определении состава ключей в данном случае, когда после нормализации не удалялись кортежи отношения, можно не анализировать значения атрибутов отношения, а оставить из списка ключевых атрибутов отношения R те ключи, которые состоят из атрибутов отношения R4. Аналогично рассуждаем, как и для отношения R, так как в отношении R4 имеются составные ключи, то проверяем его на наличие второй нормальной формы, анализируя по порядку ключ <1,4>. Анализ наличия функциональных зависимостей от части ключа 1 была выполнена ранее. Выполняем анализ наличия функциональных зависимостей атрибутов отношения R4 от части ключа 4. Таких зависимостей нет, так как атрибут 5 является атомарным ключом, а оставшиеся атрибуты отношения 1 и 2 входят с атрибутом 4 в составной ключ, отсюда также не могут от него зависеть. Осталось проанализировать составной ключ <2,4>, а более точно наличие зависимости от части ключа- атрибута 2. Анализируя отношение R4, видим, что такая зависимость есть: 2→1. Таким образом, наличие частичной зависимости показывает, что отношение R4 не находится во второй нормальной форме. В результате его нормализации получим два отношения: R5(1,2) и R6(2,4,5). На рисунке 2.16 показаны отношения R5 и R6.

Полученные отношения R5 и R6 находятся в третьей нормальной форме. Отношение R5 имеют только два атрибута, а в отношении R6 все атрибуты либо являются первичными ключами, либо входят в составной ключ. Таким образом, в результате применения описанного алгоритма нормализации исходного отношения R получена совокупность следующих схем отношений, находящихся в третьей нормальной форме: R3(3,6), R4(1,3), R5(1,2), R6(2,4,5).

R5(1,2)

1	2
1	1
2	2
3	3
4	4
5	5
2	6
6	7

R6(2,4,5)

2	4	5
1	1	1
2	2	2
3	3	3
2	4	4
4	5	5
5	2	6
6	6	7
7	7	8

Рисунок 2.16 – Отношения R5 и R6 после нормализации отношения R2

В следующем пункте рассмотрен пример выполнения анализа базы данных, модель которой рассмотрена в первом разделе, на наличие третьих нормальных форм отношений.

2.7. Пример анализа отношений базы данных на третью нормальную форму

Рассмотрим анализ базы данных «ГИБДД» на третью нормальную форму (см. рисунок 1.28). Все отношения базы данных находятся в первой нормальной форме, так как атрибуты отношений не являются множеством или списком. Отношения «Тип телефона», «Тип нарушения», «Район» находятся в третьей нормальной форме, так как количество атрибутов в отношениях меньше трех. Отношения «Адрес», «Владелец», «Нарушение» находятся во второй нормальной форме. Проверим, находятся ли отношения «Транспортное средство», «Телефон», «Нарушение_Тр.средство» во второй нормальной форме. Выполним анализ отношения «Транспортное средство» на наличие частичных функциональных зависимостей. В этом отношении один составной ключ<Номер паспорта, гос.номер>. От части ключа атрибуты, не входящие в ключ и не являющиеся атомарными ключами, не зависят. Действительно: Номер паспорта-/->Мощность двигателя, Но-

мер паспорта-/->Марка, Номер паспорта-/->Дата выпуска. Аналогично нет зависимостей и для части ключа Гос.номер. То есть отношение находится во второй нормальной форме. Аналогично в отношении «Нарушение_Тр.средство» отсутствуют зависимости атрибута, не входящего в ключ, <Код типа нарушения> от частей составного ключа <Номер паспорта, гос.номер,номер протокола>. В отношении «Телефон» отсутствуют зависимости атрибута «Код типа телефона» от частей составного ключа «Номер паспорта, номер телефона». То есть отношения «Нарушение_Тр.средство» и «Телефон» также находятся во второй нормальной форме. Отношения «Телефон» и «Нарушение_Тр.средство» находятся и в третьей нормальной форме, так как для проверки на наличие транзитивных зависимостей в отношениях кроме ключа должно быть как минимум два атрибута, а в этих отношениях имеется только один атрибут. Проверим, находятся ли в третьей нормальной форме отношения «Адрес», «Владелец», «Транспортное средство» и «Нарушение». Например, в отношении «Нарушение» имеются следующие зависимости: Номер протокола → Место нарушения -/->Дата нарушения, Номер протокола →Дата нарушения -/-> Место нарушения, то есть отсутствуют транзитивные зависимости и отношение находится в третьей нормальной форме. Аналогично в отношениях «Адрес», «Владелец», «Транспортное средство» отсутствуют связи между атрибутами, не входящими в ключ, отсюда эти отношения также находятся в третьей нормальной форме.

В заключение данного раздела еще раз отметим, что теория реляционных баз данных лежит в основе алгоритмов, реализующих операции над реляционными данными, реализации ограничений целостности данных, а также определяет правила преобразования базы данных в системах OLTP, ее нормализации для увеличения эффективности работы с базами данных в таких системах. Следует отметить, что вопросы, связанные с нормализацией отношений и приведением к той или иной нормальной форме, решаются проектировщиками баз данных с учетом целей системы. Нормализация устраняет избыточность хранения данных и нежелательные эффекты при ведении данных, но увеличивает число взаимосвязанных отношений и тем самым замедляет операции

выборки данных, выполнения запросов к базе данных. Так, в базах данных, предназначенных для систем OLAP, отношения даже не находятся в первой нормальной форме.

В следующем разделе будут рассмотрены вопросы, связанные с физической организацией баз данных, знание которых также существенно влияет на разработку баз данных и последующую эффективную работу с системами баз данных.

Глава 3. ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

3.1. Формат и размещение физических (хранимых) записей

Этап логического проектирования баз данных заканчивается определением содержания логических записей, типов элементов записи, их длины. На этапе физического проектирования осуществляется выбор формата физической записи, размещение записей в памяти и выбор метода доступа к данным. Главный принцип при определении структуры физической записи – сохранение содержания логической записи. *Хранимая запись* – совокупность связанных элементов данных, соответствующих одной или нескольким логическим записям. Хранимая запись (физическая запись) состоит из двух частей: служебной части и информационной.

Служебная часть хранимой записи используется для идентификации записи, задания ее типа, хранения признака удаления, для хранения указателей на элементы записи, идентификатора длины записи, для установления структурных ассоциаций между записями, для кодирования значений элементов. Пользовательские программы не имеют доступа к служебной части физической записи. Поля *информационной части хранимой записи* содержат значения элементов данных.

Существует несколько способов размещения элементов данных в хранимой записи, или иначе видов формата хранимой записи. Например, рассмотрим следующие виды организации хранимых записей или виды форматов хранимых записей: позиционный, с разделителями, индексный и с описателями. На рисунке 3.1 приведен пример структуры логической записи таблицы «Студент».

FIO	PREDMET	DATA	SROC	OCENKA
50, C	10, C	8 D	1 L	2, N
Смирнов Сергей Дмитриевич	Математика	3.01.95	.Y.	5

Рисунок 3.1 – Пример структуры логической записи отношения «Студент»

Виды форматов хранимых записей:

а) *позиционный способ организации хранимых записей*

В каждой записи порядок следования элементов одинаковый, тип элементов не указывается, хранится только значение. Поля имеют фиксированную длину, то есть значение элемента в каждом экземпляре записи появляется с одной и той же позиции, определенной в описании структуры файла базы данных. Все записи имеют одинаковую длину. При таком формате записи можно применять эффективные алгоритмы поиска (двоичный метод, метод золотого сечения и другие), но память используется не рационально. Если данные имеют меньшую длину, чем размер поля записи, то производится выравнивание влево или вправо, и свободные места заполняются пробелами. Количество свободных мест может быть сокращено путем использования разделителей или индексного метода.

б) *способ хранения записей с разделителем*

Запись имеет поля переменной длины и соответственно сама хранимая запись имеет переменную длину. Выбирается некоторый символ разделителя, который нигде больше при организации файлов базы данных не используется. Например, символ #. Тогда хранимая запись из примера будет выглядеть следующим образом:

Смирнов Сергей Петрович#47.5#07.02.94#Y#5.

Видно, что при таком способе организации хранимых записей память экономно расходуется, но невозможно применение быстрых алгоритмов поиска данных. Выигрываем в памяти, но проигрываем в быстродействии.

в) *индексный способ организации хранимых записей*

При данном способе для определения начала значений элементов записей используется массив указателей, размещенный в служебной части записи. На рисунке 3.2 показан пример организации такой записи.

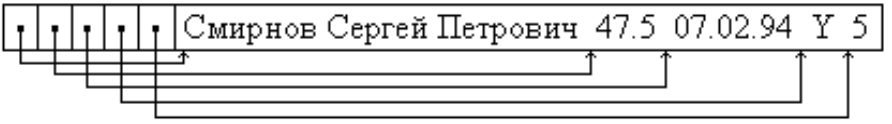


Рисунок 3.2 – Пример индексного способа организации хранимых записей

Аналогично способу хранения с разделителем индексный способ более экономно расходует память, но не позволяет для поиска данных использовать эффективные алгоритмы поиска.

г) *способ хранения с описателями*

Этот способ эффективен, если запись содержит много элементов, значения которых отсутствуют и необходимо явно хранить только элементы с известными значениями.

Например, известна фамилия студента и его средний балл, зато неизвестна оценка и признак сдачи (в срок или нет). Для этого примера введем такие описатели:

- Описатель F вместо fio = фамилия имя отчество
- Описатель P вместо predmet = средний балл
- Описатель D вместо data= дата
- Описатель R вместо Sroc = срок сдачи
- Описатель O вместо Ocenka = оценка.

Тогда хранимая запись имеет вид

F Смирнов Сергей Петрович S47.5.

Понятно, что в последних трёх способах память используется более экономно, особенно в случае отсутствия заполнения некоторых полей, зато на поиск нужной информации тратится больше времени.

Кроме формата организации хранимых записей важным является и способ кодирования конкретных значений данных. Каждый из символов элементов данных записи кодируется согласно выбранной системе кодирования в последовательность бит. Системы кодирования

могут быть разными: ASCII (американская) система, международная, полевой код и другие. Для сокращения объема памяти при размещении хранимой записи используются различные способы сжатия (например, использование аббревиатуры, подавление нулей, то есть в записи вместо шести пробелов будет цифра 6 и знак «пробел» и другие способы кодирования). Кроме того, для хранения записи могут объединяться в участки памяти фиксированного размера– это кластеризация записей.

Кластеризация записей – объединение записей различного типа в физические группы, которое позволяет как можно эффективнее использовать преимущество последовательного размещения данных. *Кластеризация* – размещение связанных данных поблизости друг от друга в целях повышения эффективности доступа. При логической кластеризации определенные типы элементов данных объединяются в записи. При физической кластеризации экземпляры записей одного или разных типов объединяются и помещаются в один блок, одну область или на одно запоминающее устройство, то есть адресуемую память фиксированного размера. Задача кластеризации состоит в определении того, каким образом для заданной модели данных необходимо хранить записи базы данных с тем, чтобы для типичных эксплуатационных нагрузок минимизировать время доступа, то есть данные наиболее часто участвующие в операциях поиска и манипулирования данными объединяются и к ним в первую очередь осуществляется обращение. В общем случае оптимальную кластеризацию трудно осуществить для сложных интегрированных баз данных, где данные, к которым осуществляется обращение с помощью различных прикладных программ, перекрываются, и легко найти компромисс между последовательными и произвольными методами поиска.

Важным понятием при рассмотрении вопроса физической организации баз данных является понятие блока. *Блок* – минимальный адресуемый элемент внешней памяти, с помощью которого осуществляется обмен информацией между оперативной и внешней памятью.

Обмен данных внешней памяти с оперативной памятью осуществляется блоками. Запись и чтение блоков осуществляется через буферную память оперативной памяти. Для организации каждого файла базы данных, в зависимости от его размера во внешней памяти, выде-

ляется от одного до N блоков, где размещаются записи (в одном блоке могут разместиться все записи или в нескольких блоках – одна запись, либо в одном блоке – одна запись; от этого будет зависеть время считывания и записи элементов файла). Каждый байт в блоке имеет свой номер: 0, 1, 2, …, x. Номер байта блока, с которого начинается запись, определяет относительный адрес записи в блоке, то есть адрес записи равен номеру блока, плюс относительный адрес в блоке (последовательный доступ), либо номер блока и значение ключа (здесь сочетание последовательного и прямого методов доступа). Записи в блоках размещаются плотно, без промежутков, последовательно одна за другой. В блоке часть памяти отводится под служебную информацию: относительный адрес свободных участков памяти, указатели на следующий блок и так далее.

Для хранения поступающих данных, которые должны размещаться в одном блоке, заполненном уже полностью, выделяется дополнительный блок памяти в области переполнения записи, организованной в виде одного блока, где записи связываются указателями в одну цепь. Обычно стремятся область переполнения ограничить, так как время поиска информации в области переполнения сильно увеличивается. Поэтому, как только достигается предел области переполнения, проводят реорганизацию файла базы данных. Файлу добавляется нужное количество блоков в основной памяти и выполняется нужная перекомпоновка записей. При этом исходят из расчета, чтобы можно было освободить область переполнения, а все записи разместить в частично незаполненных блоках (пустых или резервных для возможного добавления записей). Для каждого файла блоки пронумерованы: 1, 2, …, N и система определяет требуемый файл по имени файла и по номеру блока. Таким образом, на скорость поиска влияют: объем блока в байтах, объем файла, количество записей в блоке файла, количество записей в блоке индекса, количество блоков в файле, доля резервной части блока, число полей в записи, размер записи в байтах, длина ключевого поля в записи.

3.2. Методы доступа к данным

Метод доступа к данным – это совокупность технических и программных средств, обеспечивающих возможность хранения и выборки данных, расположенных на физических устройствах. В методе доступа важны два компонента: структура памяти и механизм поиска. *Структура памяти* – представление хранимых записей, составляющих файл, и т.д. *Механизм поиска* – алгоритм, определяющий путь доступа, который возможен в рамках заданной структуры памяти, и количество шагов вдоль этого пути для нахождения искомых данных.

Выделяют три основных группы методов доступа к данным: последовательные, индексные, произвольные методы доступа. Последовательные методы используются при поиске большого числа записей (от 10 до 100%), индексные для получения одной или нескольких записей, произвольные для получения отдельных записей.

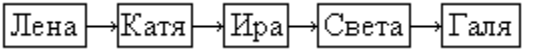
В *группе последовательных методов доступа* к данным используется один файл базы данных. Перебор записей выполняется, начиная с первой, последовательно одна за другой. Записи не упорядочены и обычно имеют фиксированный размер. В зависимости от способа организации записей в файле различают последовательный метод доступа на смежной и несмежной памяти.

а) В случае *последовательного метода доступа на смежной памяти* записи хранятся в виде логической последовательности, организация которой похожа на одномерный линейный массив. Например, в файле хранятся имена:

Лена	Катя	Ира	Света	Галя

При поиске записи, например со значением «Ира», поиск начинается с первой записи до тех пор, пока не будет найдена нужная запись или не будет достигнут конец файла. При добавлении новой записи запись добавляется в конец файла. При удалении записи сначала осуществляется её поиск, затем, начиная со следующей, записи по одной записи последовательно переписываются на предыдущее место, после чего самая последняя запись файла удаляется. Либо все записи,

начиная со следующей после удаляемой, переписываются в новый файл и все записи из него записываются в исходный файл, начиная с указателя на удаляемую запись, а конечная запись удаляется. При изменении записи сначала выполняется последовательный поиск нужной записи, затем удаляется и добавляется новая запись



б) В случае *последовательного метода доступа на несмежной памяти* записи хранятся в виде линейного списка. В данном случае невозможно использование бинарного поиска, т.к. данные не упакованы плотно и отсутствует возможность вычисления промежуточных адресов.

Все алгоритмы при работе с линейными списками хорошо изучены и реализованы. При поиске нужной записи осуществляется проход по списку, начиная с первого элемента, пока не будет найден нужный элемент списка. Для внесения изменения в некоторое информационное поле записи файла осуществляется последовательный перебор элементов списка с последующим исправлением информационного поля нужного элемента списка. При удалении, после того как нужный элемент найден, указатель на удаляемый элемент переносится на следующий за ним элемент, а от самого удаляемого элемента освобождается память. При добавлении новый элемент добавляется в конец списка.

В группе *индексных методов доступа* все действия с записями осуществляются при наличии двух файлов – индексного и основного файла базы данных. В индексном файле хранятся значения первичных ключей основного файла базы данных и адреса или указатели на записи основного файла с соответствующими значениями первичного ключа. Индексный файл всегда упорядочен по значениям ключей и, соответственно, поиск в нём осуществляется быстрыми методами поиска. Индексный файл значительно меньше собственно базы данных. В зависимости от способа организации индексного файла и основного

файла базы данных имеется множество разновидностей индексных методов доступа. Некоторые из них рассмотрены ниже.

В *индексно-последовательном методе доступа* индексный файл всегда упорядочен по первичному ключу (главному атрибуту физической записи), по значению которого идентифицируется физическая запись. Индексный файл содержит ссылки не на каждую запись базы данных, а на группу записей, хранимых в физическом блоке, по диапазону ключей. Например, если в блоке хранится 10 записей, то для него в индексном файле будет одна статья, а не 10, и размер индексного файла уменьшится в 10 раз. Таким образом, индексно-последовательный метод доступа строится на основе упорядоченного физически файла и иерархической структуры индексов блоков, каждый из которых упорядочен по значениям первичных ключей подобно записям в файле данных. Каждое значение ключа в индексе некоторого j-уровня представляет собой наибольшее значение ключа в блоке индекса или блоке данных на уровне (j + 1). На каждом уровне индекса и данных последовательный поиск выполняется до тех пор, пока не будет установлено местонахождение искомой записи или ее отсутствие. На рисунке 3.3 приведен пример организации индексно-последовательного метода доступа к данным.

При добавлении записи, например со значением ключа 103, в индексном файле одним из быстрых методов поиска ищется номер блока, в котором необходимо разместить запись с указанным ключом. В данном случае это четвёртый блок, где запись размещается между 100 и 105, так как в блоках основного файла эти записи упорядочены. При поиске нужной записи по заданному ключу сначала ищется быстрым методом поиска в индексном файле номер блока, содержащего искомую запись, а затем последовательным методом в найденном блоке основного файла осуществляется поиск требуемой записи. При удалении записи сначала она ищется описанным выше способом, а затем удаляется из блока способом, описанным в последовательном методе доступа.

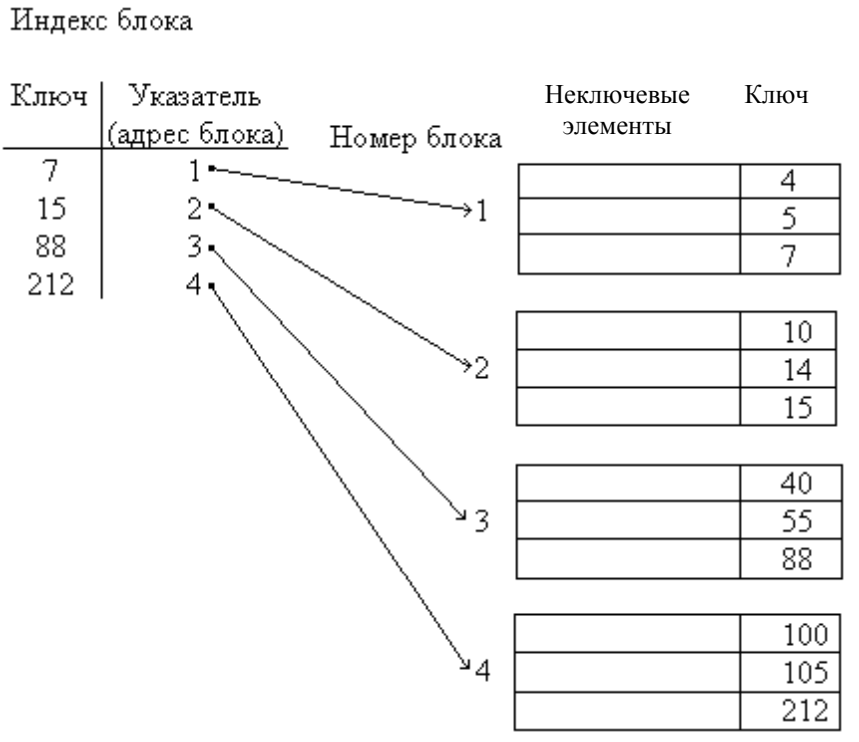


Рисунок 3.3 – Пример организации индексного файла
в индексно-последовательном методе доступа к данным

При индексно-произвольном методе доступа записи в основном файле базы данных хранятся в произвольном порядке и создается отдельный файл статей, включающих значения ключа и физические адреса хранимых записей. Статья, содержащая действительный ключ и адрес, называется *статьей индекса*, а весь файл – *индексом*. Каждой записи базы данных соответствует статья индекса. На рисунке 3.4 показан пример организации индексного файла в индексно-произвольном методе доступа к данным. Особенность этой разновидности индексного метода доступа состоит в том, что основной файл базы данных не упорядочен и адреса записей с соответствующим значением ключа индексного файла определяются одним из способов произвольных мето-

дов доступа, например с помощью функции хеширования. Организация индексного файла может быть с плотным, иногда говорят с полным, индексом (в данном случае в индексном файле хранятся ключи всех записей) и неплотным индексом, как показано на рисунке. При добавлении записи заносятся в конец основного файла и в индексном файле вставляются в соответствующее место, чтобы не нарушить упорядоченность записей. При изменении, удалении записей сначала осуществляется быстрый их поиск в индексном файле, после чего по найденному адресу записи в основном файле производятся изменения и удаления в основном файле и, если необходимо, то изменения и удаления в индексном файле.

К группе индексных методов доступа относится метод «Бинарное дерево». Запись бинарного дерева состоит из поля ключа записи и двух полей для указателей. Один из указателей хранит адрес на левое поддерево, второй указатель – на правое поддерево. Листовые указатели записи бинарного дерева содержат указатели на блоки файла основных записей (файла данных). Записи бинарного дерева содержат только одно поле данных – информационное, являющееся значением ключа. На рисунке 3.5 показан пример организации индексного файла в виде бинарного дерева.

Существуют эффективные алгоритмы работы с древовидными структурами, включая добавление, удаление, изменение узлов дерева. К индексным методам доступа относятся и метод доступа двусвязное дерево, и мультисписковый метод доступа, название которых раскрывает способ организации индексных файлов. Особое место среди индексных методов доступа занимает инвертированный метод доступа, с помощью которого осуществляется реализация поиска по вторичному ключу и не реализуются алгоритмы ведения данных.

В *инвертированном методе доступа* используется три файла – индекс вторичного ключа, инвертированный файл и основной файл базы данных. Индекс вторичного ключа содержит значения вторичных ключей, упорядоченных по возрастанию или убыванию, и адрес блока в инвертированном файле.

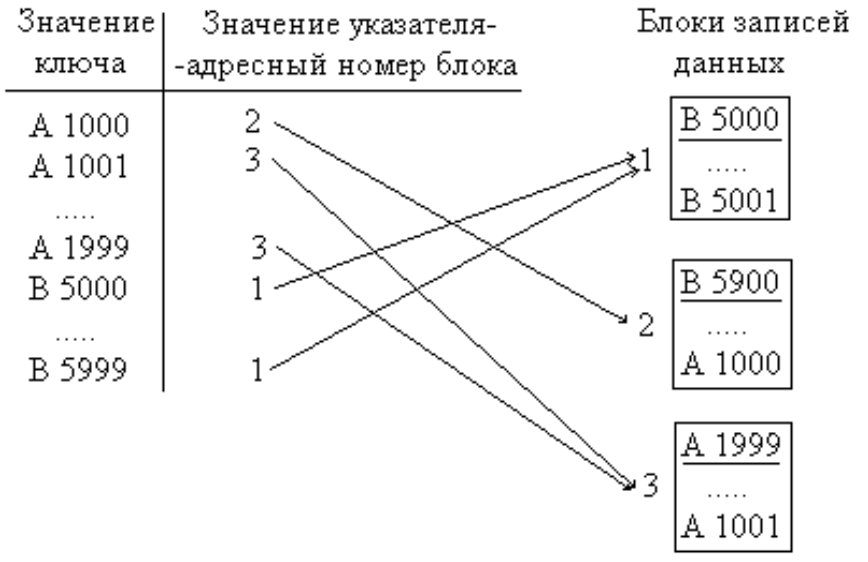


Рисунок 3.4 – Пример организации индексного файла
в индексно-произвольном методе доступа к данным

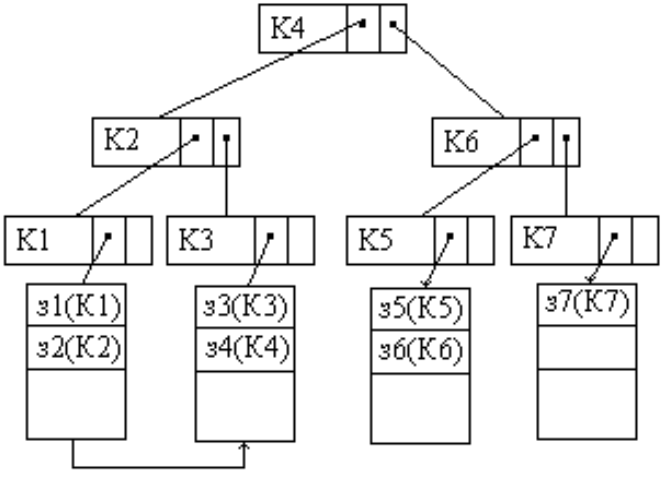
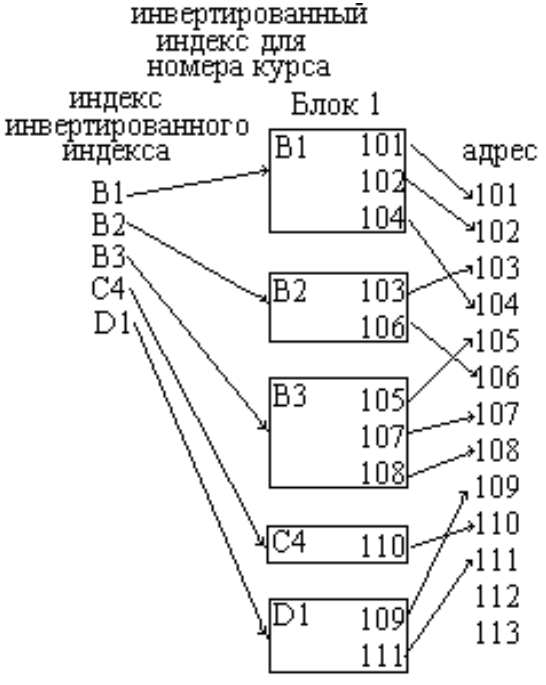


Рисунок 3.5 – Пример организации индексного файла
в виде бинарного дерева

В инвертированном файле содержатся значения первичных ключей с данным значением вторичного ключа и адреса записей в основном файле, соответствующие значениям первичного ключа. На рисунке 3.6 показан пример организации инвертированного метода доступа.

С учетом особенностей организации индексно-последовательный метод поиска используют для организации первичных ключей, индексно-произвольный метод поиска реализует альтернативные или потенциальные ключи, инвертированный метод доступа реализует инверсионные входы.

В *группе произвольных методов доступа* используются основной файл базы данных и функция, преобразующая ключ в адрес, по которому хранится в основном файле запись с соответствующим значением ключа. К произвольным методам доступа относятся прямой метод доступа и метод хеширования идентификатора.



Файл данных

фамилия студента	номер курса
Алешин	B1
Александров	B1
Борисов	B2
Волков	B1
Иванов	B3
Петров	B2
Сидоров	B3
Морковкин	B3
Гаврилов	D1
Дмитриев	C4
Козырев	
Смирнов	D1

Рисунок 3.6 – Пример организации инвертированного метода доступа

В *прямом методе доступа* каждой записи соответствует одно значение адреса. Если проектировщик базы данных может предусмотреть в памяти для каждой записи место, определяемое уникальным значением ее первичного ключа, тогда можно построить первичную функцию преобразования ключа в адрес, обеспечивающую запоминание и выборку каждой записи в точности за один произвольный доступ к блоку. С этой целью каждой совокупности ключевых значений записей, расположенных в одном и том же физическом блоке, можно присвоить относительный номер блока (относительный физический адрес). На самом деле данные хранятся в соответствии с порядком ключей, но при этом неявно им последовательно присвоены относительные номера блоков.

Например, компания имеет 30 отделений (с номерами от 1 до 30). В каждом блоке хранится 5 записей данных об отделениях, следовательно, нужно 6 блоков.

Функция преобразования ключа в адрес: $относительный\_адрес\_блока = \left\lceil \frac{номер\_отделения}{5} \right\rceil$. Например, запись данных об отделениях располагается по адресу с относительным номером 3. Такое абсолютное соответствие между ключом и относительным адресом блока является основной отличительной чертой прямого метода доступа. В случае реальных данных не всегда возможно такое соответствие значения ключа ровно одному адресу, поэтому часто приходится строить более сложные функции.

Метод хеширования идентификатора или метод рандомизации (random – произвольный доступ; hash – размешивание) – это метод быстрой выборки и обновления записей, относящийся к группе произвольных методов доступа. В этом методе используется следующая терминология. *Идентификатор* – атрибут, уникально определяющий каждый экземпляр некоторой сущности предметной области. В контексте файла и базы данных идентификатор – это первичный ключ. *Хеширование* – метод доступа, обеспечивающий адресацию данных путем преобразования значения ключа в относительный или абсолютный физический адрес. *Функция преобразования ключа* – функция хеширования или функция рандомизации. В прямом методе доступа име-

ет место отображение 1:1, в методе хеширования – M : 1, то есть возможно преобразование двух или более значений ключа в один и тот же физический адрес, так называемый *собственный адрес*. Такие ключи называют *синонимами*, а случай преобразования ключа в уже занятый собственный адрес – *коллизией*.

Для прямого метода доступа характерна «неуправляемость ключа», то есть по значению ключа один раз вычисляются адреса и размещаются в памяти, это требует неоправданно больших служебных издержек памяти.

Хеширование обеспечивает эффективную выборку и обновление отдельных записей по заданному значению первичного ключа. Плата за эффективность – нарушение упорядоченности файла и потеря возможности выполнить пакетную обработку или генерацию отчетов, основанную на упорядоченности записей по первичному ключу. Один из способов организации хеширования – это когда все адресное пространство, непосредственно доступное функции хеширования, делится на несколько областей фиксированного размера, называемых *бакетами*. В качестве бакета можно использовать цилиндр, дорожку, блок и так далее, то есть любой участок памяти, адресуемый как единое целое. Бакет может состоять из более мелких физических единиц данных (дорожка, хранимая запись и другие). Наименьшая составная единица бакета, используемая при анализе метода хеширования, называется *фрагментом* (хранимой) записи данных или секцией. Пример организации метода хеширования идентификатора показан на рисунке 3.7

Процесс разрешения коллизий синонимов состоит из двух шагов: на первом шаге выполняется просмотр бакета с целью выявления в нем свободного пространства для новой записи. При наличии пространства просмотр прекращается. В противном случае, переходим на выполнение второго шага, который состоит в обработке переполнения.

В случае заполнения области переполнения и попытке размещения очередной записи в занятый бакет происходит либо реорганизация файла (освобождается область переполнения, выделяется большее число бакетов либо размер бакетов увеличивается), либо выбирается другая функция хеширования.

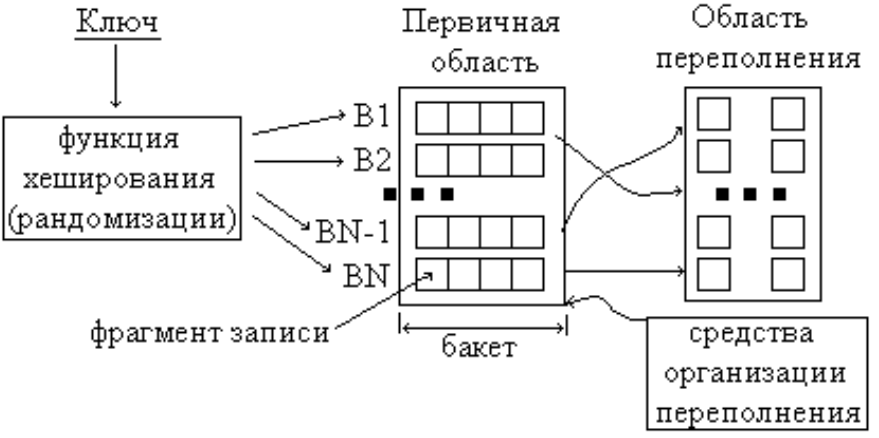


Рисунок 3.7 – Пример организации метода хеширования идентификатора

Наилучшей функцией хеширования является функция, отображающая NR значений ключа в NR собственных адресов без синонимов. Теоретически будет NR! способов такого идеального отображения. Но если учесть, что существует NR^{NR} способов присвоения NR ключами NR собственных адресов, то вероятность этого идеального отображения ничтожна. Рассмотрим пример. Пусть необходимо построить хеш-функцию, ставящую в соответствие каждому значению ключа адрес для последовательности чисел 36, 16, 13, 5, 85 и для адресного пространства из 10 бакетов, причём построить функцию хеширования, не приводящую к коллизиям:

а) $f_{хеш}$ = значение ключа (mod 10) + 1. Значения 36 и 16 синонимы и они отображаются в бакет 7, а также 5 и 85, которые отображаются в бакете 6, то есть имеется четыре коллизии;

б) $f_{хеш}$ = сумма цифр значения ключа (mod 10) + 1. Тогда получим такое размещение значений ключа по адресам:

3 + 6 = 9; 9 mod 10 + 1 = 10
$f_{хеш}(16) = (1 + 6) \bmod 10 + 1 = 8$
$f_{хеш}(13) = (1 + 3) \bmod 10 + 1 = 5$
$f_{хеш}(5) = 5 \bmod 10 + 1 = 6$
$f_{хеш}(85) = 13 \bmod 10 + 1 = 4$

То есть для данной комбинации цифр здесь нет коллизий. Метод деления является наиболее часто применяемым методом при построении функций хеширования.

Глава 4. СВОЙСТВА БАЗ ДАННЫХ

4.1. Целостность данных

Основными проблемами в процессе проектирования баз данных является обеспечение целостности, согласованности, восстанавливаемости, безопасности и эффективности, а также возможное возникновение последствий предполагаемого роста баз данных в будущем.

База данных обладает свойством целостности, если она удовлетворяет определенным ограничениям на значения и структуру данных. Возможны следующие виды *ограничений на значения* данных:

1. Данные должны находиться в заданном диапазоне. Например, зарплата служащего не может быть выше зарплаты директора или год поступления в институт меньше года рождения. Если база заполнена записями, то такой контроль ограничений значений данных осуществляется проверкой попадания в указанный диапазон и удалением некорректных записей. В дальнейшем это ограничение должно контролироваться всякий раз, когда в базу добавляется новая запись. Из смысла этого ограничения следует, что оно не должно проверяться при операции удаления записей из базы данных.

2. Данные могут принимать значения из некоторого множества значений (например, название профессий), при этом создаются специальные файлы данных (базы данных) – словари, связанные с исходным файлом, из которых без набора с клавиатуры данных вводятся нужные данные, что уменьшает возможные ошибки, учитывает ограничения на данные (должен быть машинист, а не машинистка).

3. Если поле является первичным ключом, то значение этого поля должно быть уникально. Система управления базы данных должна отклонить любую попытку ввести в базу данных кортеж (запись) со значением первичного ключа, повторяющем уже имеющееся значение.

4. Отношение может иметь кроме первичного ключа также возможные ключи (вторичные), значения которых должны быть также уникальными либо могут повторяться и использоваться для связи один ко многим. Для спецификации этого типа ограничений достаточно фразы UNIQUE (A) при описании модели данных, где A – атрибут или комбинация атрибутов.

5. В заданном отношении для каждого кортежа между значением атрибутов A и B должно всегда выполняться некоторое условие (например, что первый атрибут больше второго). Ограничение специфицируется специальным выражением.

6. Множество значений одного атрибута вычисляется через операции над другими. Реализуется с использованием функций СУБД.

Ограничения на структуру данных:

1. Контроль типа поля данных при всех операциях над базой (например, если целое поле, то заполнять, заменять и т.д. на строковое нельзя).

2. Ограничение на число полей в записи.

3. Ссылочная целостность означает, что все изменения во взаимосвязанных отношениях должны выполняться согласованно.

Основная идея обеспечения ограничений целостности заключается в том, чтобы использовать язык манипулирования данными как средство выражения этих ограничений. Описание ограничений целостности содержит две части. В первой – выражается собственно ограничение, во второй – описывается то, когда и при каких условиях должна выполняться проверка.

4.2. Свойство безопасности и секретности баз данных

Под безопасностью данных понимается защита *от непреднамеренного* доступа к данным, *идентификации* и *разрушения*. *Под секретностью данных* понимается защита от преднамеренного доступа, модификации и разрушения.

При *решении вопросов защиты* данных необходимо:

1. Определить права доступа отдельных пользователей к определенным группам данных и ограничений на характер операций, выполняемых пользователем с этими данными. Возможны следующие уровни доступа к данным:

1) неограниченный доступ ко всем отношениям для всех типов операций;
2) запрет на доступ к любым частям отношения для всех типов операций;
3) доступ к любой части отношения, но без права изменения ее содержимого (только просмотр);
4) доступ к любой части отношения, но с правом изменения значений заданного перечня атрибутов: $A_1 \ldots A_n$;
5) неограниченный доступ только к одному кортежу отношения для всех типов операций;
6) доступ только к одному кортежу отношения без права изменения содержимого этого кортежа;
7) неограниченный доступ к атрибутам $A_1A_2 \ldots A_r$ отношения для всех типов операций и запрет доступа к остальным атрибутам отношения;
8) доступ к атрибутам $A_1A_2 \ldots A_r$ отношения без права изменения их значений и запрет доступа к остальным атрибутам отношения;
9) разрешить доступ к атрибутам $A_1A_2 \ldots A_r$, применяя вычислительные операторы без права изменения их самих;
10) доступ к данным, но в ограниченном времени (от t_1 до t_2) или дате.

2. Организовать систему контроля доступа к данным.

3. Тестирование всех вновь создаваемых средств защиты данных.

4. Периодическое проведение проверок правильности функционирования данных и т.п.

Основные *методы и приемы защиты* данных:

1. Идентификация пользователя, по которой определяется уровень доступа пользователя к данным (пароли, ответы на вопросы или реализация некоторых алгоритмов).

2. Контроль и фиксация попыток нарушения допустимого уровня доступа с последующими штрафными функциями.

3. Использование специальных методов кодирования данных (один из простейших методов – перекомпоновка символов в кортеже, записи, сообщении, другой метод – замена символа или символов другим символом или группой символов этого же или другого алфавита и др.).

4. Защита данных от сбоев в аппаратных или программных средствах: хранение поколений данных (bak), формирование контрольных точек (промежуточная перезапись).

4.3. Восстанавливаемость, согласованность и эффективность баз данных

Восстанавливаемость – это возможность восстановления целостности данных после любого сбоя системы.

Информация, содержащаяся в базе данных, обычно представляет огромную ценность для ее владельца. Поэтому недопустимо, чтобы, например, механические повреждения привели к потере данных. Поэтому в СУБД должны быть средства восстановления данных после сбоев или отказов. Отказы бывают системными (физический, порча диска и т.п.) и отказы транзакций (сбой программы, т.е., например, целой переменной присвоили логическое значение) и т.д. Способы бывают следующие:

1. Специально разрешенные программы, которые сохраняют информацию о каждой транзакции (*транзакция* – единовременно выполняемая неделимая последовательность операций над данными, не нарушающая их целостность). Это приводит к тому, что в случае отката очередной транзакции результат действия предыдущего состояния данных сохранен.

2. Сохранение в буфере при выполнении транзакции содержимого неизменяемых файлов (файл – дубликат), если проведена защита от неправильной корректировки данных.

3. Фактическая корректировка данных откладывается до завершения транзакции (вместо этого заносится в отдельный файл образов).

4. Защита от системных сбоев (отказов) – это периодическое копирование информации на дублирующие носители. Информация, записанная на такой носитель, называется *дампом*.

Согласованность – свойство базы данных по отношению к некоторой совокупности пользователей, состоит в том, что в любой момент времени база данных реагирует на их запросы одинаковым образом.

Эффективность – возможность использования вычислительных ресурсов в приемлемое для пользователя время в процессе выполнения приложений базы данных, используя при этом минимальное количество внешней памяти для хранения данных.

4.4. Реорганизация баз данных. Администратор баз данных. Словарь данных

Реорганизация базы данных – изменение концептуальной, или логической, или физической структуры данных. Изменение концептуальной и логической структуры – *реконструирование*. Изменение физической структуры – *реформатирование*.

Реконструирование включает, например, добавление нового типа элементов в структуру записи, изменение атрибута записи, изменение связей между отношениями.

Реформатирование – изменение методов доступа к данным, изменение форматов записи, способов кодирования записей, изменение функции хеширования идентификатора при произвольном методе доступа к данным.

Администратор базы данных – лицо или группа лиц, осуществляющих управление в базе данных и решающих все вопросы по реорганизации базы данных. К вопросам управления в базе данных относятся:

1) определение необходимости реорганизации;
2) состав изменения;
3) способ реорганизации;
4) время проведения реорганизации;

5) определения уровня доступа к данным.

На первом этапе проектирования базы данных необходимо собрать сведения о предметной области, в том числе о назначении, способах использования и о структуре данных, а по мере развития системы необходимо накопление информации о концептуальной, логической и внутренней моделях данных, для этой цели и используется словарь данных.

Словарь данных содержит информацию об источниках, форматах и взаимосвязях между данными, их описания, сведения о характере использования и распределения ответственности. Словарь сам по себе является базой «данных о данных», руководством по базе данных.

Назначение словаря:

1) документирование данных;
2) обеспечение эффективного взаимодействия между различными категориями разработчиков и пользователей.

Словарь данных может быть автоматизированным и неавтоматизированным. Последний не может обеспечить получение по-разному отсортированных списков элементов данных, которыми пользуется разработчик. Все большее распространение получают автоматизированные словари, средства создания и ведения которых заложены в СУБД. Для эффективности словаря данных необходимо определить следующие составные элементы базы данных:

- элемент данных должен иметь уникальное имя;
- групповой элемент данных (объект-сущность, отношение);
- выводимый элемент данных (формула для его определения);
- синонимы, омонимы;
- описание концептуальной модели, логических и внешних моделей, физической модели (структура данных, методы доступа, коэффициенты блокирования).

Глава 5. ЯЗЫК SQL. СТАНДАРТ ЯЗЫКА SQL

5.1. История SQL. История стандарта SQL. Уровни соответствия. Классы инструкций SQL

В начале семидесятых годов был создан программный продукт для работы с реляционной моделью данных SEQUEL (Structured English Query Language) – структурированный английский язык для запросов, который в настоящее время называют SQL (Structured Query Language). Хотя компания IBM первая разработала теорию реляционных баз данных, первой на рынок с этой технологией вышла компания Oracle. Через некоторое время SQL завоевал на рынке популярность и привлек внимание Американского национального института по стандартизации (American National Standards Institute, ANSI), который в 1986, 1989, 1992, 1999 и 2003 годах выпустил стандарты языка SQL. Стандарт 1992 года имеет обозначение SQL92 или SQL2, стандарт 1999 года SQL99 или обозначают SQL3, сейчас используется промышленный стандарт SQL2003, который по сравнению с SQL3 имеет расширения, касающиеся объектно-ориентированных типов и базовых функций OLAP. Уровни соответствия стандарту были впервые предложены в 1992 году. Были введены три категории соответствия: начальный уровень, средний и полный (Entry, Intermediate, Full). Чтобы соответствовать стандарту, производитель SQL должен был выпустить продукт не ниже уровня Entry.

В стандарте SQL2003 существуют семь основных классов инструкций – инструкции для установления связи, управления, работы с данными, диагностики, работы со схемами, работы с сеансами, работы с транзакциями. Постоянное развитие стандарта способствовало появлению среди разных производителей и платформ многочисленных диалектов SQL. Кроме перечисленных инструкций в диалектах обычно вводятся процедурные команды (условные, циклические,

средства работы с массивами, переменными и другие). Наиболее распространены следующие диалекты: PL/SQL (Oracle), Transact/SQL (MS SQL Server), PL/pgSQL (PostgreSQL), SQLPL (DB2).

Отличительной особенностью языка SQL является тот факт, что это язык работы с множествами, представленными в реляционной форме. На основе теории множеств в терминологии ANSI используются кластеры, содержащие множество каталогов, каталоги содержат множество схем, схемы содержат множество объектов, объекты содержат подобъекты и так далее. Для доступа к подобъекту – полю таблицы нужно выполнить следующее разыменование: каталог.схема.объект.подобъект. Далее в следующих пунктах данного раздела пособия будут рассмотрены основные инструкции стандарта SQL2003, а также введены понятия основных объектов реляционных баз данных.

5.2. Идентификаторы. Константы. Операторы. Типы данных. Ограничения

В этом разделе предполагается, что основные понятия языков программирования известны, главное внимание уделено именно особенностям языка SQL.

Идентификаторы SQL имеют длину 128 символов, могут содержать любые цифры, буквы и знак подчеркивания, должны начинаться с буквы, не могут содержать пробелов и специальных символов, должны быть уникальными в своей области видимости.

Константы могут быть числовые, строковые константы, константы даты/времени и булевские. Основные категории и *типы данных* приведены на рисунке 5.1.

Операторы сравнения: = (равно), <(меньше), >(больше), <= (меньше или равно), >= (больше или равно), <>, != (не равно), !< (не меньше),!> (не больше). *Арифметические операторы*: + (сложение), – (вычитание), * (умножение), / (деление), %(остаток целочисленного деления).

Логические операторы:

ALL – возвращает true, если весь набор true;

AND – возвращает true, если все сравнения true;

ANY – возвращает true, если хотя бы одно сравнение из набора дает true;

Between – возвращает true, если операнд находится внутри диапазона;

Exists – возвращает true, если подзапрос возвращает хотя бы одну строку;

In – возвращает true, операнд равен одному выражению из списка или множеству строк, возвращаемых подзапросом;

Like – возвращает true, если операнд совпадает с шаблоном;

NOT – возвращает true, если отрицается значение булевого оператора;

OR – возвращает true, если одно из сравнений возвращает true;

SOME – возвращает true, если несколько сравнений из набора дают результат true;

IS NULL – возвращает true, если операнд имеет неопределенное значение;

NOT IS NULL –возвращает true, если операнд не имеет неопределенное значение.

Ограничение – средство, позволяющее автоматически обеспечить целостность данных, определяемое при создании таблиц. Различают четыре типа ограничений: ограничение первичного ключа (Primary Key), ограничение уникальности (Unique), ограничение внешнего ключа (Foreign Key), ограничение значением (Check). Ограничения могут приниматься на уровне столбцов и на уровне таблиц. Ограничения на уровне столбцов объявляются при создании столбца и применимы только к нему. Ограничения на уровне таблиц объявляются независимо от определения столбцов в конце инструкции создания таблиц Create Table и могут применяться к одному или нескольким столбцам таблицы. Ограничения определяются при создании таблицы или ее изменении.

<Ограничение>::= [CONSTRAINT [<имя ограничения>]]<тип ограничения> [(<имя столбца>[,…])][<предикат>].

Ограничение первичного ключа (Primary Key) объявляется для одного или нескольких столбцов, значения которых однозначно идентифицируют каждую запись таблицы. В таблице может существовать только один первичный ключ. Столбцы первичного ключа можно определить на уровне столбца или таблицы, если первичный ключ состоит из нескольких столбцов. Столбцы первичного ключа не могут содержать типы данных BLOB,CLOB,NCLOB,ARRAY. Значения первичного ключа не могут иметь значения Null.

Для составного первичного ключа комбинация значений во всех столбцах должна быть уникальна и не равна Null. Ограничение первичного ключа на уровне столбца выглядит так: <имя столбца> <тип> PIMARY KEY. Ограничение первичного ключа на уровне таблицы:

[CONSTRAINT [<имя ограничения>]] PRIMARY KEY (<имя столбца>[,…]).

Ограничение внешнего ключа (Foreign Key) объявляется для одного или нескольких столбцов, ссылающихся на столбец с ограничением уникальности, или первичного ключа другой таблицы. В одной таблице может быть несколько внешних ключей. Ограничение внешнего ключа может определяться на уровне столбца:

[CONSTRAINT [<имя ограничения>]]REFERENCES <имя связанной таблицы> [(<имя связанного столбца>)].

Ограничение внешнего ключа может задаваться на уровне таблицы:

CONSTRAINT [<имя ограничения>] FOREIGN KEY (<имя связанного столбца> [,…])REFERENCES <имя связанной таблицы> [(<имя связанного столбца>[,…])].

Ограничение уникальности (UNIQUE) является ограничением альтернативного ключа или потенциального ключа, то есть имеет уникальное значение, но не является первичным ключом. Ограничение уникальности на уровне столбца:

<имя столбца> <тип> UNIQUE.

Категория	Примеры типов данных и аббревиатура	Описание
BINARY () двоичные	BINARY LANGE OBJECT (BLOB)	Хранятся двоичные строковые значения в шестнадцатеричном формате. Двоичные строки хранятся без ссылок на наборы и без ограничения длины
BOOLEAN (булевы)	BOOLEAN	Хранятся сведения об истинности – TRUE или FALSE
CHARACNER Строковые типы (символьные)	CHAR CHARACNER VARYNG(VARCHAR)	Хранятся любые символы и строки фиксированной длины. Вариабельные типы хранят строки переменной длины, при этом автоматически удаляются замыкающие пробелы. Другие типы пробелы оставляют
	NATIONAL CHARACTER (NCHAR) NATIONAL CHARACNER VARYNG (NVARCHAR)	Типы данных, связанные с национальными символами, разработаны для поддержки конкретных наборов символов, зависящих от реализации
	CHARACTER LANGE OBJECT (CLOB)	Строковые типы больших размеров
	NACIONAL CHARACTER LANGE OBJECT (NCLOB)	Строковые типы больших размеров с поддержкой конкретных наборов символов, зависящих от реализации
DATALINK (Связь с данными)	DATALINK	Определяет ссылку на файл или другой внешний источник
INTERVAL (интервал)	INTERVAL	Определяет набор временных значений или промежуток времени
COLLECTION (коллекция)	ARRAY MULTISET	Отсортированные коллекции фиксированной и несортированные коллекции данных переменной длины
NUMERIC (числовые)	INTEGER (INT) SMALLINT BIGINT NUMERIC(p,c) DEC[IMAL](p,c) FLOAT(p,c) REAL DOUBLE PRECISTION	Типы INT, BIGINT, SMALLINT хранят точные числовые значения с заранее заданной точностью (precision,p) и масштабом (scale,s), равным нулю. Типы NUMERIC,DEC хранят точные числовые значения с регулируемой точностью и масштабом. FLOAT хранит числовые значения с плавающей точкой с регулируемой точностью
TEMPORAL (время)	DATE TIME TIME WITH TIME ZONE	Хранит дату или дату и время. Может включать сдвиг, соответствующий часовому поясу
XML	XML	Хранит данные формата XML

Рисунок 5.1 – Основные категории и типы данных SQL2003

Ограничение уникальности на уровне таблицы:

[CONSTRAINT [<имя ограничения>]] UNIQUE
(<имя столбца>[,…]).

Ограничение значением (CHEK) выполняет сравнение значения поля, для которого введено ограничение, с определенным условием. Ограничение значением на уровне поля определяется следующим образом:

[CONSTRAINT [<имя ограничения>]] CHEK (<условие>).

5.3. SQL – инструкции для работы со схемами

Схема – это объект базы данных, являющийся набором объектов базы данных или, иначе, это контейнер объектов. К основным объектам реляционной базы данных относятся таблицы, хранимые процедуры и функции, виды, триггеры, пользовательские типы, пользовательские роли, схемы. Для создания объекта любого вида используется команда CREATE, для изменения объектов – команда ALTER, для удаления объектов – команда DROP. Команда создания таблиц выглядит следующим образом:

CREATE TABLE <имя таблицы> (<имя поля> <тип поля> [<
ограничения на уровне поля>], […] ,<ограничение на уровне
таблицы> […]).

Хранимая процедура – объект базы данных, являющийся набором откомпилированных команд. Хранимые процедуры используются для ускорения выполнения набора команд, для передачи параметров, обеспечения модульной структуры приложений. Команда создания хранимой процедуры выглядит следующим образом:

CREATE PROCEDURE <имя процедуры> [(<описание списка
параметров>)] [<опции>] AS <тело процедуры>.

Хранимая функция – хранимая процедура, возвращающая одно значение, сохраняемое в имени функции. Команда создания хранимой функции выглядит следующим образом:

CREATE FUNCTION <имя функции> [(<описание списка параметров>)] RETURNS <тип функции> [<опции>] AS <тело функции>.

Вид (представление) – объект базы данных, представляющий собой виртуальный набор данных, созданный на основе выборки из базы данных. Вид создается для обеспечения защиты данных, для сокращения программного кода при выборке данных. Виды могут быть *простыми*, если они созданы на основе одной таблицы, *составными*, если они созданы на основе нескольких таблиц, и *видами из видов*, если при их создании используются ранее определенные виды. Команда создания вида выглядит следующим образом:

CREATE VIEW <имя вида> [<опции>] AS <команда выборки данных>.

Триггер – специальный вид хранимой процедуры, автоматически срабатывающий при выполнении команд ведения данных (добавления, удаления, изменения). Триггер создается для обеспечения сложных ограничений целостности данных. Команда создания триггера выглядит следующим образом:

CREATE TRIGGER <имя триггера> [<вид триггера>] <команды, вызывающие срабатывание триггера> ON <имя таблицы> AS <тело триггера>.

Пользовательский тип – объект базы данных, определяющий ограничения на уровне домена данных, созданный на основе стандартного типа данных. Команда создания пользовательского типа выглядит следующим образом:

CREATE TYPE <имя типа> AS <имя стандартного типа> [<атрибуты>].

Пользовательская роль – набор поименованных прав пользователей. Пользовательская роль используется для реализации в базах данных системы безопасности. Команда создания пользовательской роли выглядит следующим образом:

CREATE ROLE <имя роли> AS <список разрешений>.

Команда создания схемы данных выглядит следующим образом:

CREATE SCHEMA <имя схемы> AS <список объектов>.

Каждая команда работы с объектами имеет в различных диалектах языка SQL свои синтаксические особенности.

5.4. SQL – инструкции для работы с данными

К инструкциям для работы с данными в реляционных базах данных относят команды добавления, удаления, изменения, выборки, команды работы с курсорами. *Команда добавления* выглядит следующим образом:

INSERT [INTO] {<имя таблицы>|<имя вида>} [(<список полей>)] {VALUES (<список значений>) [(список значений), …]|DEFAULT VALUES|<команда выборки данных>|<команда вызова хранимой процедуры>}.

Особенности выполнения команды добавления и всех команд ведения данных рассмотрены в разделе операций обновления данных (пункт 2.1). *Команда изменения* данных выглядит следующим образом:

UPDATE {<имя таблицы>|<имя вида>} [<псевдоним>]
SET <имя поля>=<значение> [<имя поля>=<значение>, …]
[WHERE<условие>|WHERE CURRENT OF <имя курсора>].

Команда удаления выглядит следующим образом:

DELETE FROM <имя таблицы>[WHERE <условие>|WHERE CURRENT OF <имя курсора>].

Команда выборки данных, реализующая операцию селекции, выглядит следующим образом:

SELECT [<ограничения на количество выводимых записей>]
<список выбираемых элементов>
FROM <откуда осуществляется выборка>
[WHERE <условие выборки>]
[GROUP BY <список группируемых элементов>[HAVING <условие на группируемые данные>]]
[ORDER BY <список сортируемых элементов>].

В разделе команды SELECT <ограничения на количество выводимых записей> могут присутствовать следующие ключевые слова: ALL (означающее выборку всех записей из полученного набора данных, без учета наличия возможных повторений), DISTINCT (выборка неповторяющихся записей из набора), TOP <количество > [PERCENT] (выборка указанного количества записей, возможно не записей, а числа записей в процентном отношении от общего количества записей выборки).

В разделе команды SELECT <список выбираемых элементов> указывается через запятую список отбираемых элементов набора данных. Элементом может быть поле таблицы, может быть скалярная функция или функция агрегирования, может быть подзапрос. Элементу можно присвоить новое имя или псевдоним, которое видно только в пределах самой команды. Для введения псевдонима используется синтаксическая конструкция: <элемент> AS <псевдоним>.

В разделе команды SELECT <откуда осуществляется выборка> создается реляционный набор данных. В этом разделе указывается список элементов, включающий таблицы, виды, подзапросы, которые могут быть соединены с помощью операций соединения. Каждый элемент этого раздела также можно переименовать, введя псевдоним. Соединение элементов осуществляется следующим образом:

<элемент> [[AS] <псевдоним>] [<тип соединения>] JOIN
<элемент> [[AS] <псевдоним>] ON <условие соединения>.

Тип соединения может быть внутренним (INNER), левым внешним (LEFT OUTER), правым внешним (RIGHT OUTER), полным (FULL). Особенности этих соединений были рассмотрены в пункте 2.1.

В разделе команды SELECT <условие выборки> присутствует предикат, сформированный с помощью операций отношений, логических операций, скалярных функций с применением имен элементов присутствующих в разделах <список выбираемых элементов> и <откуда осуществляется выборка>.

В разделе команды SELECT <список группируемых элементов> указывается список имен элементов из раздела <список выбираемых элементов> или номеров этих элементов. В результате выполнения этого раздела команды SELECT данные набора объединяются (группируются), возможно, в несколько множеств записей, каждое из которых имеет одинаковые значения элементов, присутствующих в списке группируемых элементов. В сочетании с группировкой обычно используется выполнение функций агрегирования для каждого множества сгруппированных данных. При необходимости ввести ограничение на функции агрегирования используется раздел <условие на группируемые данные>.

В разделе команды SELECT <список сортируемых элементов> указываются через запятую имена или номера элементов из раздела <список выбираемых элементов>. В результате выполнения этого раздела команды данные в результате выборки сортируются в указанном порядке, насколько это возможно в реляционной модели данных.

Подзапрос – запрос, вложенный в основной запрос. Подзапрос синтаксически, внутри основного запроса, заключается в круглые скобки. Подзапросы могут встречаться в разных разделах инструкции SELECT. Различают следующие виды подзапросов. *Скалярный подзапрос* – подзапрос, возвращающий одно значение. *Табличный подзапрос* – подзапрос, возвращающий несколько значений. *Вложенный табличный подзапрос* – подзапрос, возвращающий несколько столбцов и несколько строк. Каждый из названных подзапросов могут участвовать в коррелированных подзапросах. *Коррелированный подзапрос* – подзапрос, зависящий от значения во внешнем запросе. То есть в коррелированном запросе внутренний запрос выполняется по одному разу для

каждой записи, извлеченной внешним запросом. Если существует несколько уровней вложенности подзапросов, коррелированный подзапрос может ссылаться на любой уровень главного запроса, который выше его собственного уровня. Схематично разделы команды SELECT с точки зрения видов допустимых подзапросов можно представить таким образом:

SELECT (<скалярный подзапрос>) FROM (<вложенный табличный подзапрос>) AS<псевдоним> WHERE <нечто>=(<скалярный подзапрос>)|<нечто> IN (<табличный подзапрос>) |EXISTS (<любой вид подзапроса>).

Курсор – временный набор данных, созданный на основе выборки данных. В отличие от объектов базы данных, курсор постоянно не хранится в базе данных, а используется во время сеанса работы пользователя с базой данных. Курсор необходим для доступа к отдельным записям полученного набора данных. Работа с курсором включает следующие этапы: объявление курсора, открытие курсора, доступ к отдельным записям курсора, закрытие курсора и освобождение памяти от курсора. В стандарте языка SQL для объявления временных данных, в том числе переменных, курсоров, используется команда DECLARE. Для открытия курсора команда OPEN, для доступа к отдельной записи команда FETH, для закрытия курсора команда CLOSE, для освобождения памяти команда DEALLOCATE. В диалектах SQL синтаксис перечисленных команд различается, как и наличие различных видов курсоров.

5.5. SQL – инструкции для работы с сеансами

Для изменения настроек сеанса работы пользователя с базой данных в языке SQL используется команда установки SET. Эта команда имеет одну из следующих синтаксических конструкций:

SET <ключевое слово> ON|OFF
SET <ключевое слово> <опции>
SET <переменная>=<значение>.

Таких команд установки в каждом диалекте обычно бывает много для реализации настройки сеанса пользователя. Например, для присвоения сеансу конкретной или текущей роли используется команда SET ROLE {<имя роли>|NONE}. Для определения уровня изолированности пользователей используется команда:

SET TRANSACTION ISOLATION LEVEL <опции>.

5.6. SQL – инструкции для работы с транзакциями

Транзакция – неделимая с точки зрения воздействия на базу данных последовательность операций, завершающаяся фиксацией или откатом. Главное назначение транзакций – это сохранение базы данных в целостном состоянии при выполнении различных операций. Для создания транзакции используется команда BEGIN TRANSACTION. Для выполнения отката транзакции, в результате чего база данных возвращается в исходное состояние, используется команда ROLLBACK. Для фиксации в базе данных сделанных в транзакции операций используется команда COMMIT. В случае необходимости реализации логики внутри транзакции могут быть использованы точки сохранения транзакций – SAVEPOINT TRANSACTION. Понятие транзакции является фундаментальным в задачах систем с распределенной обработкой данных, основы которых будут рассмотрены в следующей главе.

Глава 6. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАСПРЕДЕЛЕННЫХ БАЗ ДАННЫХ

6.1. Основные понятия систем с распределенной обработкой данных

Распределённая база данных – это база данных, состоящая из нескольких, возможно пересекающихся или дублирующих друг друга частей, хранимая в различных ЭВМ вычислительной сети. Распределённая база данных находится на рабочих станциях вычислительной сети.

Основная задача систем управления распределёнными базами данных состоит в обеспечении средства интеграции локальных баз данных, располагающихся в некоторых узлах вычислительной сети, с тем чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных как к единой базе. При этом должны обеспечиваться: простота использования системы, возможности автономного функционирования при нарушениях связности сети или при административных потребностях, высокая степень эффективности.

В настоящее время системы коллективного доступа к данным могут поддерживать одну из следующих трех моделей: модель файлового сервера; сервера базы данных; сервера приложений. Модель файлового сервера представлена на рисунке 6.1.

При реализации модели файлового сервера приложение выполняется на рабочих станциях, где находится и ядро СУБД, база данных расположена на сервере. Такие системы имеют низкую производительность, так как промежуточные данные передаются по низкоскоростной шине сети, а прикладные программы и ядро СУБД находятся на маломощных рабочих станциях. Такую модель в системах коллективного доступа к данным реализуют фактически локальные СУБД Fox-

Pro, Access,Clipper,Paradox,dBase и другие. Модель файлового сервера не поддерживает распределенную обработку данных.

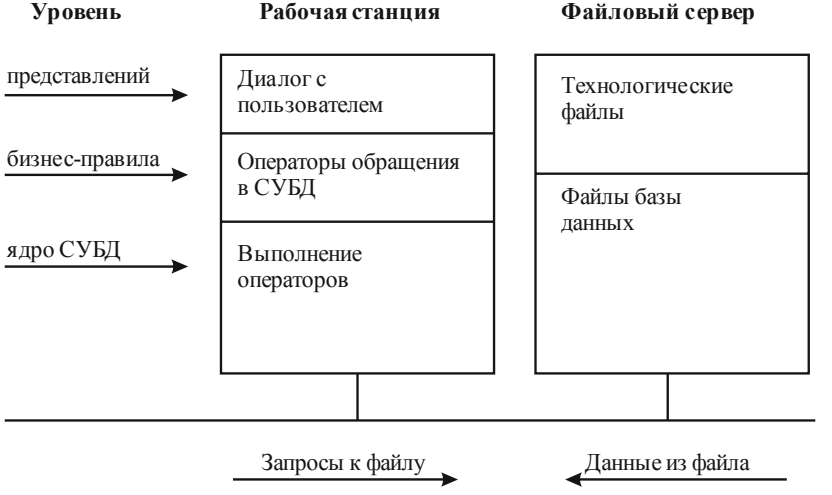


Рисунок 6.1 – Модель файлового сервера

На рисунке 6.2 приведена модель сервера базы данных.

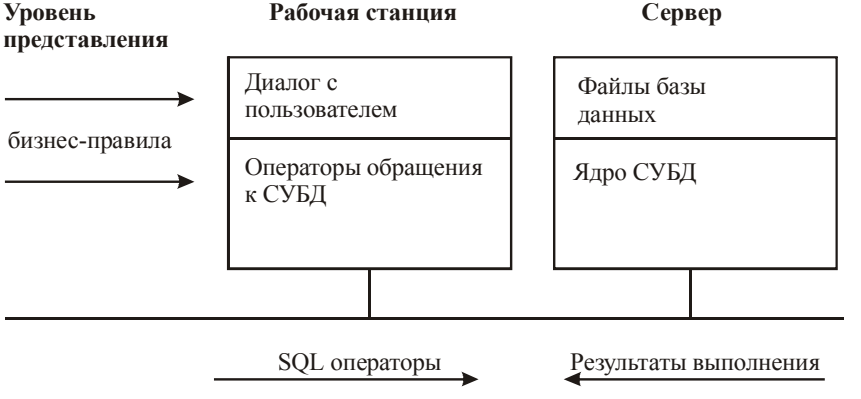


Рисунок 6.2 – Модель сервера базы данных

При реализации модели сервера базы данных приложение выполняется на рабочих станциях. Операторы не выполняются на рабо-

чих станциях, а пересылаются для обработки на сервер. Ядро СУБД транслирует запрос и выполняет его, обращаясь к индексам и другим промежуточным данным. Обратно на рабочую станцию передаются только результаты обработки оператора. Эта модель поддерживает распределенную обработку данных. Модель сервера базы данных поддерживают такие СУБД, как Oracle, MS SQL Server, Sybase, Informix, Ingress и другие. Системы с такой моделью имеют высокую производительность, так как по шине передаются только SQL-запросы и результаты выполнения. Сами запросы выполняются на высокоскоростных серверах. Недостатком такой модели является использование дорогих сервера и сети. На рисунке 6.3 показана модель сервера приложений. Сервер приложений можно организовать с помощью хранимых процедур на языках, таких как PL/SQL, TRANSACT SQL и других. Эта модель также поддерживает распределенную обработку данных. Системы на базе моделей сервера базы данных и сервера приложений поддерживают архитектуру клиент-сервер.

Сервер базы данных в системах распределенной обработки – это логический процесс, отвечающий за обработку запросов к базе данных.

Рисунок 6.3 – Модель сервера приложений

Клиент базы данных – это логический процесс, посылающий серверу запрос на обслуживание. Архитектура клиент-сервер позволяет нескольким клиентам использовать совместно один сервер. При большом числе клиентов такая архитектура приводит к увеличению

вероятности отказов ее компонентов. Клиент также выполняет функции представления интерфейса пользователя и может выполнять несложные операции обработки данных. Другие функции распределены между несколькими компьютерами – серверами баз данных и сервером приложений. Серверы баз данных управляют данными, а сервер приложений выполняет все вычисления, связанные с реализацией бизнес – логики приложения. Такую архитектуру называют многозвенной.

Существуют различные виды распределенных баз данных. Возможны однородные и неоднородные распределённые базы данных. В однородном случае каждая локальная база данных управляется одной и той же СУБД. В неоднородной системе локальные базы данных – это актуальная, но очень сложная проблема. Многие решения известны на теоретическом уровне, но пока не удаётся справиться с главной проблемой – недостаточной эффективностью интегрированных систем. Более успешно практически решается промежуточная задача – интеграция неоднородных SQL-ориентированных систем. Понятно, что этому в большой степени способствуют стандартизация языка SQL и общее следование производителей СУБД принципам открытых систем.

6.2. Изолированность пользователей в многопользовательских системах

В многопользовательских системах с одной базой данных одновременно могут работать несколько пользователей или прикладных программ. Задачей системы является обеспечение изолированности пользователей, то есть создание достоверной и надёжной иллюзии того, что каждый из пользователей работает с базой данных в одиночку. В связи со свойством сохранения целостности базы данных транзакции являются подходящими единицами изолированности пользователей. Действительно, если с каждым сеансом работы с базой данных ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием базы данных, то есть с таким состоянием, в котором база данных могла бы находиться, даже если бы пользователь

работал с ней в одиночку. При соблюдении обязательного требования поддержания целостности базы данных возможны следующие уровни изолированности транзакций:

Первый уровень изолированности пользователей – отсутствие потерянных изменений. Здесь будем считать, что пользователь или транзакция, запускаемая пользователем, одно и то же. Рассмотрим следующий сценарий совместного выполнения двух транзакций. Транзакция первая изменяет объект базы данных A. До завершения первой транзакции, вторая транзакция, как и первая транзакция, изменяет тот же объект базы данных A, и вторая транзакция завершается оператором ROLLBACK (например, по причине нарушения ограничений целостности). Тогда, при повторном чтении объекта базы данных, транзакция 1 не видит изменений этого объекта, произведённых ранее. Такая ситуация называется ситуацией потерянных изменений и противоречит требованию изолированности пользователей. Во избежании такой ситуации для первой транзакции требуется, чтобы до ее завершения никакая другая транзакция не могла изменять объект A. Отсутствие потерянных изменений является минимальным требованием к СУБД по части синхронизации параллельно выполняемых транзакций.

Второй уровень изолированности пользователей – отсутствие чтения «грязных данных». Рассмотрим следующий сценарий совместного выполнения транзакций 1 и 2. Транзакция 1 изменяет объект базы данных A, параллельно с этим транзакция 2 читает тот же объект базы данных. Поскольку операция изменения ещё не завершена, транзакция 2 видит несогласованные «грязные» данные. Это тоже не соответствует требованию изолированности пользователей, так как каждый пользователь начинает свою транзакцию при согласованном состоянии базы данных и ожидает увидеть согласованные данные. Чтобы избежать ситуации чтения «грязных» данных, до завершения транзакции 1, изменившей объект A, никакая другая транзакция не должна читать объект A (необходимо выполнить блокировку чтения объекта A до завершения операции его изменения в транзакции 1).

Третий уровень изолированности пользователей – отсутствие неповторяющихся чтений. Рассмотрим следующий сценарий. Транзакция 1 читает объект базы данных A, и до ее завершения транзакция

2 изменяет объект A и успешно завершается оператором COMMIT. Транзакция 1 повторно читает объект A и видит его изменённое состояние. Чтобы избежать неповторяющихся чтений, до завершения транзакций 1 никакая другая транзакция не должна изменять объект A. В большинстве систем это является максимальным требованием к синхронизации транзакций, хотя отсутствие неповторяющихся чтений ещё не гарантирует реальной изолированности пользователей.

Существует возможность обеспечения разных уровней изолированности для разных транзакций, выполняющихся в одной системе баз данных. Существует ряд приложений, для которых первого уровня изолированности пользователей достаточно (например, прикладные и системные утилиты, для которых некорректность индивидуальных данных несущественна). При этом удаётся существенно сократить накладные расходы СУБД и повысить общую эффективность.

К более тонким проблемам изолированности транзакций относится так называемая проблема *кортежей – "фантомов" (четвертый уровень изолированности пользователей)*, вызывающая ситуации, которые также противоречат изолированности пользователей. Рассмотрим следующий сценарий. Транзакция 1 выполняет оператор выборки кортежей отношения R с условием выборки S (то есть выбирается часть кортежей *отношения* R, удовлетворяющих условию S). До завершения транзакции 1 транзакция 2 вставляет в отношение R новый кортеж r, удовлетворяющий условию S, и успешно завершается. Транзакция 1 повторно выполняет оператор A и в результате появляется кортеж, который отсутствовал при первом выполнении оператора (фантом). Такая ситуация также противоречит идее изолированности транзакций и может возникнуть даже на третьем уровне изолированности транзакций. Чтобы избежать появления кортежей–фантомов, требуется более высокий «логический» уровень синхронизации транзакций. Идеи такой синхронизации (предикатные синхронизационные захваты) известны давно, но в большинстве систем не реализованы.

6.3. Сериализация транзакций. Методы сериализации транзакций

Для того чтобы добиться изолированности транзакций, в СУБД необходимы методы регулирования совместного выполнения транзакций. *План* (способ) *выполнения набора транзакций* называется *сериальным*, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций. *Сериализация транзакций* – это механизм их выполнения по некоторому сериальному плану. Обеспечение такого механизма является основной функцией компонента СУБД, ответственного за управление транзакциями. Система, в которой поддерживается сериализация транзакций, обеспечивает реальную изолированность пользователей. Основная реализационная проблема состоит в выборе метода сериализации набора транзакций, который не слишком ограничивал бы их параллельность. Тривиальным решением является действительно последовательное выполнение транзакций. Но существуют ситуации, в которых можно выполнять операторы разных транзакций в любом порядке с сохранением сериальности. Примерами могут служить только читающие транзакции, а также транзакции, не конфликтующие по объектам базы данных. Между транзакциями могут существовать следующие виды конфликтов: W-W– транзакция 2 пытается изменять объект, изменённый не закончившейся транзакцией 1; R-W– транзакция 2 пытается изменять объект, прочитанный не закончившейся транзакцией 1; W-R– транзакция 2 пытается читать объект, изменённый не закончившейся транзакцией 1. Практические методы сериализации транзакций основываются на учёте этих конфликтов.

Существует два базовых подхода к сериализации транзакций – один основан на синхронизационных захватах объектов базы данных, второй на использовании временных меток. Суть обоих подходов состоит в обнаружении конфликтов транзакций и их устранении. Следует заметить, что для каждого из подходов имеются две разновидности методов – пессимистическая и оптимистическая. При применении пессимистических методов, ориентированных на ситуации, когда конфликты возникают часто, конфликты распознаются и разрешаются немедленно при их возникновении. Оптимистические методы

основываются на том, что результаты всех операций модификации базы данных сохраняются в рабочей памяти транзакций. Реальная модификация базы данных производится только на стадии фиксации транзакций. Тогда же проверяется, не возникают ли конфликты с другими транзакциями. Пессимистические методы сравнительно просто трансформируются в свои оптимистические варианты.

Методы, основанные на синхронизационных захватах, в свою очередь делятся на методы: двухфазного протокола синхронизационных захватов; гранулированных синхронизационных захватов; предикатных захватов.

Сначала рассмотрим *метод двухфазного протокола синхронизационных захватов*.

Метод состоит в том, что перед выполнением любой операции в транзакции над некоторым объектом базы данных от имени транзакции запрашивается синхронизационный захват требуемого объекта в соответствующем режиме (в зависимости от вида операции). Основными режимами синхронизационных захватов являются:

Совместный режим – S (Shared), означающий разделяемый захват объекта и требуемый для выполнения операции чтения объекта;

Монопольный режим – X (eXclusive), означающий монопольный захват объекта и требуемый для выполнения операции занесения, удаленя и модификации.

Захваты объектов несколькими транзакциями по чтению совместимы, то есть нескольким транзакциям допускается читать один и тот же объект, захват объекта одной транзакцией по чтению не совместим с захватом другой транзакцией того же объекта по записи. Захваты одного объекта разными транзакциями изображены на следующей таблице:

Текущее состояние объекта	X	S
Нет захвата –	Да	Да
X	Нет	Нет
S	Нет	Да

В первом столбце приведены возможные состояния объекта с точки зрения синхронизационных захватов. При этом «–» соответствует состоянию объекта, для которого не установлен никакой захват. Транзакция, запросившая синхронизационный захват объекта БД, уже захваченный другой транзакцией в несовместимом режиме, блокируется до тех пор, пока захват с этого объекта не будет снят. Заметим, что слово «нет» в таблице соответствует описанным ранее возможным случаям конфликтов транзакций по доступу к объектам базы данных (WW, RW, WR). Совместимость S– захватов соответствует тому, что конфликт RR не существует.

Для обеспечения сериализации транзакций (третьего уровня изолированности) синхронизационные захваты объектов, произведённые по инициативе транзакций, можно снимать только при её завершении. Это требование порождает двухфазный протокол синхронизационных захватов – 2PL. В соответствии с этим протоколом выполнение транзакции разбивается на две фазы: первая фаза транзакции – накопление захватов; вторая фаза (фиксация или откат)– освобождение захватов. При соблюдении двухфазного протокола синхронизационных захватов действительно обеспечивается сериализация транзакций на третьем уровне изолированности. Основная проблема состоит в том, что следует считать объектом для синхронизационного захвата. В контексте реляционных баз данных возможны следующие виды захватываемых объектов:

файл – физический объект, область хранения нескольких отношений и, возможно, индексов;

отношение – логический объект, соответствующий множеству кортежей данного отношения;

страница данных – физический объект, хранящий кортежи одного или нескольких отношений, индексную или служебную информацию;

кортеж – элементарный технический объект базы данных. На самом деле, когда мы говорим про операции над объектами базы данных, то любая операция над кортежем фактически является и операцией над страницей, в которой этот кортеж хранится, и над соответствующим отношением, и над файлом, содержащим отношение.

Поэтому действительно имеется выбор уровня объекта захвата. Чем крупнее объект синхронизационного захвата (неважно, какой природы этот объект– логический или физический), тем меньше синхронизационных захватов будет поддерживаться в системе и на это, соответственно, будут тратиться меньшие накладные расходы. Более того, если выбрать в качестве уровня объектов для захватов файл или отношение, то будет решена даже проблема фантомов (если это не ясно сразу, посмотрите ещё раз на формулировку проблемы фантомов и определение двухфазного протокола захватов). Но при использовании для захватов крупных объектов возрастает вероятность конфликтов транзакций и тем самым уменьшается допускаемая степень их параллельного выполнения. Разработчики многих систем начали с использования страничных захватов, полагая это некоторым компромиссом между стремлениями сократить накладные расходы и сохранить достаточно высокий уровень параллельности транзакций. Но это не очень хороший выбор. Использование страничных захватов в двухфазном протоколе вызывает неприятные синхронизационные проблемы, усложняющие организацию СУБД. В большинстве современных систем используются покортежные синхронизационные захваты. Но если единицей захвата является кортеж, то какие синхронизационные захваты потребуются при выполнении таких операций, как уничтожение отношения? Подобные рассуждения привели к разработке аппарата гранулированных синхронизационных захватов.

При применении *метода гранулированных синхронизационных захватов* захваты могут запрашиваться по отношению к объектам разного уровня: файлам, отношениям и кортежам. Требуемый уровень объекта определяется тем, какая операция выполняется (например, для выполнения операции уничтожения отношения объектом синхронизационного захвата должно быть всё отношение, а для выполнения операции удаления кортежа – этот кортеж). Объект любого уровня может быть захвачен в режиме S или X. Вводится специальный протокол гранулированных захватов и новые типы захватов: перед захватом объекта в режиме S или X соответствующий объект, более верхнего уровня, должен быть захвачен в режиме IS, IX или SIX.

Захват IS (Intented for Shared lock) по отношению к некоторому составному объекту O означает намерение захватить некоторый входящий в O объект в совместном режиме. Например, при намерении читать кортежи из отношения R это отношение должно быть захвачено в режиме IS (а до этого в таком же режиме должен быть захвачен файл).

Захват IX (Intented for eXclusive lock) по отношению к некоторому составному объекту O означает намерение захватить некоторый входящий в O объект в монопольном режиме. Например, при намерении удалять кортежи из отношения R это отношение должно быть захвачено в режиме IX (а до этого в таком же режиме должен быть захвачен файл).

Захват SIX (Shared, Intented for eXclusive lock) по отношению к некоторому составному объекту O означает совместный захват всего этого объекта с намерением впоследствии захватывать какие–либо входящие в него объекты в монопольном режиме. Например, если выполняется длинная операция просмотра отношения с возможностью удаления некоторых просматриваемых кортежей, то экономичнее всего захватить это отношение в режиме SIX (а до этого захватить файл в режиме IS).

Довольно трудно описать словами все возможные ситуации. Ниже приведена таблица совместимости захватов:

	X	S	IX	IS	SIX
–	Да	да	да	да	да
X	Нет	нет	нет	нет	нет
S	Нет	да	нет	да	нет
IX	Нет	нет	да	да	нет
IS	Нет	да	да	да	да
SIX	Нет	нет	нет	да	нет

Несмотря на привлекательность метода гранулированных синхронизационных захватов, следует отметить, что он не решает пробле-

му фантомов (если, конечно, не ограничиться использованием захватов отношений в режимах S и X). Для решения этой проблемы необходимо перейти от захватов индивидуальных объектов базы данных к захвату условий (предикатов), которым удовлетворяют эти объекты. Проблема фантомов не возникает при использовании для синхронизации уровня отношений именно потому, что отношение как логический объект представляет собой неявное условие для входящих в него кортежей. Захват отношения – это простой и частный случай предикатного захвата.

Поскольку любая операция над реляционной базой данных задаётся некоторым условием (то есть в ней указывается не конкретный набор объектов базы данных, над которыми нужно выполнить операцию, а условие, которому должны удовлетворять объекты этого набора), идеальным выбором было бы требовать синхронизационный захват в режиме S или X именно этого условия. Но если посмотреть на общий вид условия, допускаемых, например, в языке SQL, то становится непонятно, как определить совместимость двух предикатных захватов. Ясно, что без этого использовать предикатные захваты для синхронизации транзакций невозможно, а в общей форме проблема неразрешима. Но эта проблема сравнительно легко решается для случая простых условий.

В методе *предикатных синхронизационных захватов* рассматриваются простые условия. Условие простое, если оно является конъюнкцией простых предикатов, имеющих вид

Имя атрибута {=, >,<} значение.

Для простых условий совместимость предикатных захватов легко определяется на основе следующей геометрической интерпретации. Пусть R отношение с атрибутами $a_{1,...}a_{2,...,}a_{n,}$ а $m_{1,}$ $m_{2,...,}$ m_n – множество допустимых значений $a_1,a_{2,...,}a_n$ соответственно (все эти множества – конечные). Тогда можно сопоставить R конечное n-мерное пространство возможных значений кортежей R. Любое простое условие “вырезает”m–мерный прямоугольник в этом пространстве (m<=n). Тогда S–X, X–S, X–X предикатные захваты от разных транзакций совместимы, если соответствующие прямоугольники не пересекаются.

Главным недостатком всех методов, основанных на синхронизационных захватах, является *наличие тупиков между транзакциями*, необходимость их определения и разрушения. Например, тупик возникает между транзакциями T1 и T2 в следующем случае: транзакции T1 и T2 установили монопольные захваты объектов r1 и r2 соответственно; после этого T1 требуется совместный захват r2, а T2–совместный захват r1. Ни одна из транзакций не может продолжаться, следовательно, монопольные захваты не будут сняты, а совместные – не будут удовлетворены. Поскольку тупики возможны и никакого естественного выхода из тупиковой ситуации не существует, то эти ситуации необходимо обнаружить и искусственно устранить. Основой обнаружения тупиковых ситуаций является построение (или постоянное поддержание) графа ожидания транзакций.

Граф ожидания транзакций – это ориентированный двудольный граф, в котором существуют два типа вершин – вершины, соответствующие этим транзакциям, и вершины, соответствующие объектам захвата. В этом графе существует дуга, ведущая из вершины–транзакции к вершине–объекту, если для этой транзакции существует удовлетворённый захват объекта. В графе существует дуга из вершины–объекта к вершине–транзакции, если транзакция ожидает удовлетворения захвата объекта. Легко показать, что в системе существует ситуация тупика, если в графе ожидания транзакций имеется хотя бы один цикл. Для распознавания тупика периодически производится построение графа ожидания транзакций (как уже отмечалось, иногда граф ожидания поддерживается постоянно) и в этом графе ищутся циклы. Традиционной техникой (для которой существует множество разновидностей) нахождения циклов в ориентированном графе является *редукция графа*.

Редукция состоит в том, что, прежде всего, из графа ожидания удаляются все дуги, исходящие из вершин–транзакций, в которые не входят дуги из вершин–объектов. (Это как бы соответствует той ситуации, что транзакции, не ожидающие удовлетворения захватов, успешно завершились и освободили захваты). Для тех вершин–объектов, для которых не осталось входящих дуг, но существуют исходящие, ориентация исходящих дуг изменяется на противоположную ориента-

цию (это моделирует удовлетворения захватов). После этого снова срабатывает первый шаг, и так до тех пор, пока на первом шаге не прекратится удаление дуг. Если в графе остались дуги, то они обязательно образуют цикл. После того как найден цикл, в графе ожидания транзакций необходимо разрушить найденный тупик.

Разрушение тупика начинается с выбора в цикле транзакций так называемой транзакции–жертвы, то есть транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций. Критерием выбора является стоимость транзакции, при этом жертвой выбирается самая дешёвая транзакция. Стоимость транзакции определяется на основе многофакторной оценки, в которую с разными весами входят время выполнения, число накопленных захватов, приоритет. После выбора транзакции–жертвы выполняется откат этой транзакции, который может носить полный или частичный характер. При этом освобождаются захваты и может быть продолжено выполнение других транзакций. Такое насильственное устранение тупиковых ситуаций является нарушением принципа изолированности пользователей, которого невозможно избежать. Причем в централизованных системах стоимость построения графа ожидания сравнительно невелика, но она становится слишком большой в по-настоящему распределённых СУБД. Поэтому в таких системах обычно используются другие методы сериализации транзакций.

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций, основан на использовании временных меток.

Основная идея *метода временных меток,* у которого существует множество разновидностей, состоит в следующем: если транзакция T1 началась раньше транзакции T2, то система обеспечивает такой режим выполнения, как если бы T1 была целиком выполнена до начала T2. Для этого каждой транзакции т предписывается временная метка t, соответствующая времени начала T. При выполнении операции над объектом r транзакция T помечает его своей временной меткой и типом операции (чтение или изменение). Перед выполнением операции над объектом r транзакция T1 выполняет следующие действия: проверяет,

не закончилась ли транзакция T, пометившая этот объект. Если T закончилась, T1 помечает объект r и выполняет свою операцию. Если транзакция T не завершилась, то T1 проверяет конфликтность операций. Если операции неконфликтны, при объекте r остаётся или проставляется временная метка с меньшим значением и транзакция T1 выполняет свою операцию. Если операции T1 и T конфликтуют, то при t(T)>t(T1) (т.е. транзакция T является более «молодой», чем T1) производится откат T и T1 продолжает работу. Если же t(T)<t(T1) (T «старше» T1), то T1 получает новую временную метку и начинается заново.

К недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо. Кроме того, в распределённых системах не очень просто вырабатывать глобальные временные метки с отношением полного порядка (это отдельная большая наука). Но в распределённых системах эти недостатки окупаются тем, что не нужно распознавать тупики, а как мы уже отмечали, построение графа ожидания в распределённых системах стоит очень дорого.

6.4. Журнализация и буферизация изменений в базах данных

Одним из основных требований к развитым СУБД является надёжность хранения баз данных. Это требование предполагает, в частности, возможность восстановления согласованного состояния базы данных после любого рода аппаратных и программных сбоев. Очевидно, что для выполнения восстановлений необходима некоторая дополнительная информация. В подавляющем большинстве современных реляционных СУБД такая избыточная дополнительная информация поддерживается в виде журнала изменений базы данных. Общей целью журнализации изменений базы данных является обеспечение возможности восстановления согласованного состояния базы данных после любого сбоя. Поскольку основой поддержания целостного состояния

базы данных является механизм транзакций, журнализация и восстановление тесно связаны с понятием транзакции. Общими принципами восстановления являются следующие: результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных; результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных. Это, собственно, и означает, что восстанавливается последнее по времени согласованное состояние базы данных. Возможны следующие ситуации, при которых требуется производить восстановление состояния базы данных.

Индивидуальный откат транзакций. Тривиальной ситуацией отката транзакции является её явное завершение оператором ROLLBAK. Возможны также ситуации, когда откат транзакции инициируется системой. Примерами могут быть возникновение исключительной ситуации в прикладной программе (например, деление на ноль) или выбор транзакции в качестве жертвы при обнаружении синхронизационного тупика. Для восстановления согласованного состояния базы данных при индивидуальном откате транзакции нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции.

Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой). Такая ситуация может возникнуть при аварийном выключении электрического питания, при возникновении неустранимого сбоя процессора (например, срабатывании контроля оперативной памяти). Ситуация характеризуется потерей той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.

Восстановление после поломки основного внешнего носителя базы данных (жёсткий сбой). Эта ситуация при достаточно высокой надёжности современных устройств внешней памяти может возникать сравнительно редко, но тем не менее, СУБД должна быть в состоянии восстановить базу данных даже и в этом случае. Основой восстановления является архивная копия и журнал изменения базы данных.

Во всех трёх случаях основой восстановления является избыточное хранение данных. Эти избыточные данные хранятся в журнале, содержащем последовательность записей об изменении базы данных.

Возможны два основных варианта ведения журнальной информации. В первом варианте для каждой транзакции поддерживается отдельный локальный журнал изменений базы данных этой транзакции. Локальные журналы используются для реализации индивидуальных откатов транзакций. Кроме того, поддерживается общий журнал изменений базы данных, используемый для восстановления состояния базы данных после мягких и жёстких сбоев. Такой подход позволяет быстро выполнить индивидуальные откаты транзакций, но приводит к дублированию информации в локальных и общих журналах. Поэтому чаще используется второй вариант – поддержание только общего журнала изменений базы данных, который используется и при выполнении индивидуальных откатов. Журнализация отношений тесно связана не только с управлением транзакциями, но и с буферизацией страниц базы данных в оперативной памяти. По причинам объективно существующей разницы в скорости работы процессоров и оперативной памяти и устройств внешней памяти буферизация страниц базы данных в оперативной памяти – единственный реальный способ достижения удовлетворительной эффективности СУБД.

Если бы запись об изменении базы данных, которая должна поступить в журнал при выполнении любой операции модификации базы данных, реально немедленно вносилась во внешнюю память, это привело бы к существенному замедлению работы системы. Поэтому записи в журнал тоже буферизируются: при нормальной работе очередная страница выталкивается во внешнюю память журнала только при полном заполнении записями. Но реальная ситуация является более сложной. Имеются два вида буферов – буфер журнала и буфер страниц оперативной памяти, которые содержат связанную информацию. И те, и другие буфера могут выталкиваться во внешнюю память. Проблема состоит в выработке некоторой общей политики выталкивания, которая обеспечивала бы возможности восстановления состояния базы данных после сбоев. Проблема не возникает при индивидуальных откатах транзакций, поскольку в этих случаях содержимое оперативной памяти не утрачено и можно использовать как буфер журнала, так и буферы страниц базы данных. Но если содержимое буферов утрачено, для проведения восстановления базы данных необходимо иметь неко-

торое согласованное состояние журнала и базы данных во внешней памяти. Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц базы данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем изменённый объект оказывается во внешней памяти базы данных. Соответствующий протокол журнализации (и управления буферизацией) называется Write Ahead Log(WAL) – «пиши сначала в журнал», и состоит в том, что если требуется вытолкнуть во внешнюю память изменённый объект базы данных, то перед этим нужно гарантировать выталкивание во внешнюю память журнала записи об его изменении. Другими словами, если во внешней памяти базы данных находится некоторый объект базы данных, по отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции. Обратное неверно, то есть если во внешней памяти журнала содержится запись о некоторой операции изменения объекта базы данных, то сам изменённый объект может отсутствовать во внешней памяти базы данных. Дополнительное условие на выталкивание буферов накладывается тем требованием, что каждая успешно завершившаяся транзакция должна быть реально зафиксирована во внешней памяти. Какой бы сбой не произошёл, система должна восстановить состояние базы данных, содержащее результаты всех зафиксированных к моменту сбоя транзакций.

Глава 7. ПРИМЕР РЕАЛИЗАЦИИ РАСПРЕДЕЛЁННЫХ БАЗ ДАННЫХ. MS SQL SERVER

7.1. Основные характеристики MS SQL Server. Системные базы данных, таблицы и хранимые процедуры. Базы данных и файлы

СУБД MS SQL Server – это распределенная СУБД, реализующая клиент-серверную технологию. Задача сервера – управление хранением данных и осуществление контроля целостности. Задача клиента – осуществление контроля вводимых данных, управление интерфейсом пользователя, приведение данных к нужному формату для пользователя. Язык – Transact SQL**,** являющийся расширением языка SQL (позволяет создавать хранимые процедуры, триггеры, такие объекты баз данных, как пользовательские типы, правила, значения по умолчанию, средства работы с транзакциями). Операционная система Windows NT. СУБД MS SQL Server позволяет определять до 32767 баз данных, внутри каждой базы данных до 2 миллиардов таблиц. В каждой таблице до 250 столбцов; нет ограничения на количество строк в таблице. Для каждой таблицы можно определить до 250 индексов.

Основные службы:

1) SQL Server Management Studio – главная утилита администрирования и работы с данными;

2) SQL Server Agent – служба автоматизации административных задач;

3) SQL Server Browser – обозреватель, предоставляющий клиентским компьютерам информацию о соединениях с SQL Server;

4) SQL Server Profiler – утилита управления отчетами, генерируемыми службами.

При работе SQL Server возникает задача сохранения информации о том, как хранятся данные в базе. Информация о хранении дан-

ных называется метаданными, которые хранятся в словаре данных. Словарь данных реализуется в виде системных таблиц и заполняется при выполнении команд Create и Alter.

Базы данных делятся на системные и пользовательские базы данных. В *системных базах данных* хранятся данные, используемые системой для управления другими данными. В SQL Server имеет четыре вида системных баз данных: master, model, tempdb, msdb.

Master – системный каталог, хранящий следующую информацию: учётные записи пользователей, текущие процессы, сообщение о системных ошибках, сведения о базах данных на сервере, размер каждой базы, активные блокировки, доступные устройства баз данных для каждого пользователя, доступные резервные (dump) устройства, системные процедуры для администратора.

Model – шаблон для вновь создаваемых на сервере баз данных SQL Server создаёт копию базы данных model. В model добавляют объекты, которые используются в каждой вновь создаваемой базе данных.

Tempdb – системная база данных, которая обеспечивает место для размещения на диске временных таблиц и других объектов временного хранения, таких как курсоры, результаты группировки данных и другие. Для доступа к этой базе не надо специальных разрешений и она удаляется при прерывании связи с SQL Server.

Msdb – поддерживает службу SQL Server Agent – планировщика задач (оповещения и задания, данные об операторах) и управления ошибочными ситуациями, в эту базу входят таблицы – sysalerts – таблица о всех определённых пользователем событиях, Sysoperators – информация о всех операторах и другое при работе с интернет и электронной почтой.

Каждая системная и все пользовательские базы данных имеют системные таблицы, куда заносится информация о SQL Server в целом и о каждой базе данных в отдельности. Все системные таблицы начинаются с SYS.

Каждая *пользовательская база данных* состоит из двух и более файлов, каждый из которых может использоваться лишь одной базой. У файлов существует два имени – логическое и физическое.

Файлы делятся на три типа: *первичные файлы* (используются для хранения данных и информации, определяющих начальные действия с базой, база данных содержит лишь один первичный файл – primary, стандартное расширение- .mdf); *вторичные файлы* – secondary (одна или несколько областей для хранения данных, могут использоваться для распределения операций чтения/записи по нескольким дискам, стандартное расширение – ndf); *файлы журналов* – log (содержат журналы транзакций базы данных, база данных содержит хотя бы один файл журнала транзакций, стандартное расширение – ldf). Каждый файл может принадлежать лишь одной базе данных. Каждая база данных содержит, по крайней мере, один первичный файл и один файл журнала.

Файловая группа – это способ создания набора файлов. Файл может относиться только к одной группе (файлы журналов не могут входить в файловые группы). Файловые группы используются для распределения нагрузки, связанной с выполнением чтения и записи, по нескольким дискам для таблиц, которые размещены на этой файловой группе. Если группа содержит больше одного файла, чтение и запись для этой группы распределяется между файлами групп. Каждая база данных имеет первичную файловую группу под названием Primary. Эта группа содержит первичный файл данных и все остальные файлы данных, для которых специально не указано, что они относятся к другим файловым группам. Данные могут размещаться в любой файловой группе. Все данные, для которых специально не указано, что они будут храниться в определенной файловой группе, будут размещены в файловой группе по умолчанию. Любую файловую группу можно сделать файловой группой по умолчанию в любой момент времени. Файл и файловая группа могут использоваться только одной базой данных. Файл может быть членом только одной файловой группы. Вы должны хранить данные и записи журнала в разных файлах. Файлы данных и файлы журнала не могут принадлежать к одной файловой группе, так как файлы журнала вообще не могут входить в какие-либо файловые группы. В одной базе данных может быть одновременно до 256 файловых групп. Для повышения производительности при работе с базой данных необходимо придерживаться следующих рекомендаций:

- файлы и файловые группы размещаются по возможно большему числу локальных дисков (не рекомендуется использовать сетевые диски, так как обращение к ним происходит медленнее, чем к локальным, и быстродействие снижается);
- объекты, увеличивающиеся в размерах наиболее быстро, желательно размещать в различных файловых группах;
- таблицы, которые обычно используются вместе в запросах, размещайте в различных файловых группах, что позволит увеличить производительность за счет распараллеливания операций обращения к дискам;
- таблицы, к которым обращаются наиболее часто, и их некластерные индексы размещайте в различных файловых группах, так как при этом увеличивается производительность за счет распараллеливания операций обращения к дискам;
- не размещайте файлы журналов на тех же физических дисках, где размещены файлы и файловые группы или другие файлы, не относящиеся к SQL Server.

Журнал транзакций используется для последовательной записи изменений всех изменений, вносимых в базу данных. Главная цель журнала – обеспечение целостности данных в базе, даже если какая-то транзакция не была завершена. После перезагрузки сервера произойдет откат всех незавершенных транзакций и изменения будут отменены. В результате данные в базе данных вернутся к тому же состоянию, в котором они находились до начала незавершенной транзакции. При успешном завершении транзакции записи помечаются в журнале как завершенные и отката назад уже не произойдет.

Журнал транзакций состоит из множества виртуальных файлов журнала. Виртуальный файл журнала – это сегмент файла журнала. Когда журнал транзакций обрезается, при этом удаляется один или более виртуальных файлов журнала. Файл журнала состоит из двух или более виртуальных файлов. Минимальный размер виртуального файла журнала – 256 Кбайт. SQL Server создает новые виртуальные файлы, когда журнал увеличивается в размере. Количество и размер виртуаль-

ных файлов зависят от размера приращения (growth increment) для файла журнала. Больший размер приращения определяет и большой размер виртуальных файлов. Создание базы данных осуществляют либо через главную утилиту администрирования, используя инструментальные средства, либо с помощью команды CREATE DATABASE.

```
CREATE DATABASE <имя базы данных>
[ ON [PRIMARY] ( [NAME = <логическое имя файла>, ]
[ , FILENAME = <имя файла ОС>
[, SIZE =< размер>]
[ MAXSIZE = {<максимальный размер>| UNLIMITED}]
[ FILEGROWTH = <приращение> ] ) |
{ FILEGROUP <имя группы файлов> FILEDEFINITIONS}
[ , …n ] ]
[ LOG ON { [ NAME = <логическое имя файла> ,}
[FILENAME = <имя файла ОС>]
[, SIZE =< размер> ]
[, MAXSIZE = {<максимальный размер> | UNLIMITED}]
[, FILEGROWTH = <приращение> ] )
[ , …n ]
[FOR LOAD| FOR ATTACH].
```

В случае, если при создании базы данных не указан первичный файл данных и/или файл журнала, то отсутствующий файл (или файлы) создается с именем по умолчанию. Физические файлы будут находиться в стандартном каталоге. Первичному файлу приписывается имя: имя базы.mdf, а файлу журнала – имя базы.ldf. Если размер файлов не задан, то при создании размер первичного файла совпадает с размером первичного устройства базы данных model, а размер файла журнала и вторичных файлов равен 1 Мбайт. Он может быть и больше, если размер первичного файла базы данных model превышает 1 Мбайт. Хотя имена и размеры файлов указывать не обязательно, на практике это нужно делать. SQL Server создает базу в два этапа. На первом этапе база данных model копируется в новую базу данных, а на втором этапе инициализируется все неиспользуемое пространство.

Команда CREATE DATABASE имеет следующие параметры:

- PRIMARY – файл определяется как первичное устройство;
- NAME – логическое имя, по умолчанию совпадающее с базой данных;
- FILENAME – полное имя файла на диске;
- SIZE – исходный размер файла. Минимальный размер файла журнала равен 512 Кбайт;
- MAXSIZE – максимальный размер файла;
- UNLIMITED – размер файла не ограничивается;
- FILEGROWTH – приращение файлов в мегабайтах (MB), килобайтах (KB) или процентах (%). По умолчанию приращение равно 10%;
- FOR LOAD – обеспечивает обратную совместимость со сценариями SQL Server, написанными для предыдущих версий;
- FOR ATTACH – указывает, что файлы базы данных уже существуют.

Пользователь, создавший базу данных, является её владельцем. Все параметры конфигурации базы данных копируются из базы данных model, если только при создании базы данных не был указан параметр FOR ATTACH. Удаление базы данных осуществляется командой

- DROP DATABASE <имя базы данных> [,…n].

При удалении базы удаляются файлы, используемые ими. Нельзя удалить базу, используемую в данный момент. После удаления базы данных её файлы не могут быть использованы.

Изменение владельца базы данных осуществляется с помощью системной хранимой процедуры:

- Sp_changedbowner [@loginame=] ‘учетная запись пользователя для входа’[,[@map=]флаг уничтожения псевдонима].

Владельцем базы данных можно сделать любую учетную запись, которая в настоящий момент не является пользователем базы данных. Только пользователи с правами системного администратора имеют право менять собственника базы данных. Невозможно изменить собственника системных баз данных. Перед изменением собственника базы

данных Вы должны быть подключены к этой базе данных. Использование хранимой процедуры показано на примере:

```
USE MyDB
Go
Sp_changedbowner @loginame='NewDBO'
go.
```

Для смены имени используется хранимая процедура

Sp_renamedb «старое имя», «новое имя».

Базу данных может переименовать только системный администратор. Предварительно база данных должна быть переведена в однопользовательский режим.

Изменение определений базы данных и её размеров осуществляется командой:

- ALTER DATABASE <имя базы данных>
- {ADD FILE <спецификация> [,…n] [TO FILEROUP <имя группы>]
- | ADD LOG FILE <спецификация> [,…n]
- | REMOVE FILE <логическое имя файла>
- | ADD FILEGROUP <имя группы>
- | REMOVE FILEGROUP <имя группы>
- | MODIFY FILE <спецификация>
- | MODIFY FILEGROUP <имя группы> < свойство группы>
- }
- <спецификация> ::=
- (NAME = <логическое имя файла>
- [,FILENAME= <имя файла ОС>]
- [, SIZE = <размер>]
- [, MAXSIZE = {<максимальный размер> | UNLIMITED}]
- [, FILEGROWTH = <приращение>]).

Данная команда позволяет включать файлы в существующие группы или добавлять в базу данных новые файлы журналов. При добавлении нового файла данных без указания группы файл включается в первичную группу. Команда выполняет также удаление пустых файлов. Первичная группа всегда должна содержать как минимум один первичный файл и один файл журнала в базе данных. При удалении

файла удаляется физический файл. Осуществляет добавление новых групп файлов. При добавлении новой группы в нее не включаются никакие файлы. Осуществляет удаление групп файлов. Если в группе присутствуют какие-либо файлы, они должны быть пустыми; в противном случае удаление группы невозможно. Удалить первичную группу файлов невозможно.

Используя команду ALTER DATABASE для модификации файлов, необходимо указывать логическое имя файла. Спецификация позволяет изменить все остальные параметры, но в одной команде ALTER DATABASE можно указать один параметр. Если изменяется размер файла, новый размер не может быть меньше текущего.

При модификации группы файлов можно задавать свойства READONLY, READWRITE, DEFAULT. Свойство READONLY говорит о том, что группа файлов доступна только для чтения. Оно предотвращает возможную модификацию объектов из этой группы. Для первичной группы установка этого свойства невозможна. Свойство READONLY может устанавливаться только пользователями, обладающими монопольным доступом к базе данных. Свойство READWRITE обладает противоположным действием. Оно разрешает обновление объектов в группе. Свойство READWRITE может устанавливаться только пользователями, обладающими монопольным доступом к базе данных. Свойство DEFAULT говорит о том, что группа назначается стандартной группой для базы данных. При установке свойства DEFAULT это свойство отменяется для группы, которая была назначена стандартной ранее. При создании базы стандартной группой назначается первичная группа файлов. Если в командах CREATE NABLE, ALTER TABLE, CREATE INDEX не указана группа файлов, новые таблицы и индексы создаются в стандартной группе. Непосредственное перемещение файлов из одной группы в другую невозможно. Для этого нужно сначала удалить файл, а затем включить его в новую группу.

Уменьшение размера баз данных вручную выполняется командой

DBCC SHRINKDATABASE (<имя базы> [,< процент>]
[, { NOTRUNCATE| TRANCATEONLY}]).

При сжатии базы данных можно указать необязательный параметр – процент свободного места, которое вы хотите оставить в базе данных. Параметр NOTRUNCATE отменяет стандартное поведение и оставляет свободное место в файлах операционной системы после сжатия файла. Параметр TRANCATEONLY возвращает операционной системе все неиспользуемое место в файлах данных и сокращает файл до последнего экстента. Команда не пытается переместить записи в невыделенные страницы. При наличии параметра NOTRUNCATE значение «процент» игнорируется. Параметр NOTRUNCATE не позволяет сократить файл ниже минимального размера. В целом команда не сжимает файл больше минимума, необходимого для хранения данных в файле. Если параметр NOTRUNCATE используется вместе с параметром «процент», то последствия ограничиваются перемещением используемых страниц в начало файла. База данных не может быть меньше базы model.

Многие параметры конфигурации SQL Server настраиваются на уровне баз данных с помощью системной хранимой процедуры:

Sp_dboption [‘база данных’] [, ‘имя параметра’] [, ‘значение’].

Например, параметр:

OFFLINE – если его значение равно true, то база данных находится в отключенном состоянии и не может никем использоваться;

READ ONLY – если его значение равно true, пользователи могут только читать данные из базы, не модифицируя её;

SINGLE – задает одно- или многопользовательский режим работы системы.

Просмотр информации о базе данных, её конфигурации:

Sp_helpdb [имя базы].

Просмотр сведений о файлах, связанных с текущей базой данных:

Sp_helpfile [[@filename=]’имя ’].

Просмотр информации о группе файлов текущей базы данных:

Sp_helhfilegroup [[@filegroupname=]’имя’].

7.2. Таблицы баз данных. Создание, удаление, изменение

Команда создания таблиц соответствует стандарту языка SQL. Имена таблиц ограничиваются 128 символами. Они должны быть уникальными по отношению к владельцу. Таблица может содержать до 1024 столбцов. Имя столбца имеет длину до 128 символов и уникально в пределах таблицы (допускаются в имени служебные символы _, @, #). Типы полей также соответствуют стандарту. В качестве одного из видов полей может использоваться счетчик. Столбец счетчика (IDENTITY) – это автоматизированный столбец, генерирующий уникальные значения с некоторым приращением. Столбцы счетчика не могут содержать неопределенные значения и должны относиться к числовым типам данных – int, smallint, tinyint, numeric(p,0),decimal(p,0). Если при объявлении столбца не указаны начальное значение и приращение, столбец действует как счетчик с начальным значением 1 и приращением 1. Недостаточный размер счетчика может вызвать проблемы. В примере тип tinyint допускает максимальное значение 255, при достижении максимума таблица будет отвергать все дальнейшие операции вставки.

При изменении созданной таблицы к числу возможных модификаций относится добавление столбцов, изменение типов данных в существующих столбцах, объявление первичных ключей и так далее. Нельзя удалить ограничение первичного ключа при наличии ограничения внешнего ключа, ссылающегося на него. Команда изменения выглядит следующим образом:

- ALTER TABLE <имя таблицы> <параметры>.

Добавить столбцы после определения таблицы можно с помощью команды:

- ALTER TABLE <имя таблицы> ADD <имя поля> <тип поля> <ограничения>.

Удаление столбцов осуществляется командой:

- ALTER TABLE <имя таблицы> DROP COLUMN < имя поля>.

Удаление из таблиц ограничений:

- ALTER TABLE <имя таблицы> DROP CONSTRAINT <имя ограничения>.

Команда удаления таблицы выглядит следующим образом:

- DROP TABLE <имя таблицы>.

Удаленная таблица навсегда пропадает из базы данных, то есть действие команды удаления невозможно отменить. Удаляются не только пользовательские, но и системные таблицы. Таблицу, на которую ссылаются какие-либо ограничения, удалить нельзя. Перед удалением таблицы эти ограничения необходимо отключить или удалить. Таблица может быть удалена только владельцем. Удаленную таблицу нельзя восстановить. Кроме рассмотренных пользовательских таблиц и системных таблиц, существуют временные таблицы. Имена системных таблиц начинаются с префикса sys. Временные таблицы отличаются от обычных тем, что они не предназначены для постоянного хранения данных. Временные таблицы могут быть локальными и глобальными. Локальные временные таблицы доступны только одному пользователю во время сеанса, глобальные временные таблицы доступны всем пользователям сети во время сеанса. Имена временных таблиц начинаются на # (локальные временные таблицы) и на ##(глобальные временные таблицы). Имена временных таблиц ограничиваются 116 символами. Временные таблицы удаляются при отключении пользователя от базы данных. Временные таблицы используются так, как будто они входят в текущую базу данных, однако в действительности данные хранятся в TEMPDB. Временные таблицы удаляются как обычные таблицы во время сеанса работы пользователя.

7.3. Индексы баз данных

В SQL Server существуют индексы двух типов: *кластерные и некластерные*. Одна из особенностей кластерного индекса заключается в том, что один элемент индекса соответствует *странице* данных. Страница представляет собой физический блок для хранения данных объемом 8 Кбайт. После того как сервер определит страницу, он находит на странице нужную запись. Кластерный индекс реализует механизм ин-

дексно-последовательного доступа, причем индекс построен в виде бинарного дерева. После того как сервер определит страницу, он находит на странице нужную запись. Поскольку кластерный индекс позволяет выйти на первую запись результата и организовать последующий перебор таблицы в поисках остальных записей без возврата на уровень индекса, кластерные индексы отлично подходят для выборки интервалов данных. К этой категории относятся внешние ключи (объединения), списки имен и любые другие совокупности данных с последовательными ключами. Благодаря структуре кластерных индексов поиск в них обычно выполняется быстрее, чем при выборке произвольных записей для соответствующих некластерных индексов. Но это условное правило, зависящее от количества указателей, необходимых для индекса каждого типа, а также от их ширины. Иногда кластерный индекс работает значительно быстрее, а иногда вообще не дает выигрыша в скорости, хотя для большинства запросов кластерные индексы работают быстрее некластерных.

Некластерный индекс реализует механизм индексно-произвольного метода доступа. Его важнейшая особенность состоит в том, что один элемент индекса приходится на одну *запись*. Каждой записи в таблице назначается идентификатор. Поскольку в некластерном индексе каждый указатель соответствует одной записи, а также потому, что их указатели имеют больший размер, чем у кластерных индексов (поскольку содержат идентификатор записи, а не просто идентификатор страницы), некластерные индексы обычно занимают намного больше места. Для повышения быстродействия кластерный индекс следует создавать раньше некластерных. Создание кластерных индексов требует физической сортировки записей, а некластерные индексы содержат указатели на страницы и записи, которые приходится модифицировать при перемещении записей. (При создании кластерного индекса понадобится свободное место в базе данных в объеме отсортированной копии с индексом, или примерно 120-150% объема таблицы).

Рекомендуется помнить:

- выбор индекса определяется соображениями быстродействия, а не первичным ключом или другими обстоятельствами;

- для оптимизации физического размещения данных желательно иметь сгруппированный индекс в большинстве таблиц;
- с ростом количества индексов (больше 5) затраты становятся чрезмерными;
- в системах OLTP с оперативной обработкой транзакций создается минимальное количество индексов, что позволяет ускорить выполнение команд UPDATE, INSERT, DELETE.

Команды ведения данных при работе с таблицами соответствуют стандарту языка SQL.

7.4. Программирование на Transact SQL. Комментарии. Переменные. Команды управления

Пакет – группа команд, передаваемых от клиента серверу. Пакет может содержать одну или несколько команд. Команды пакета анализируются, компилируются и выполняются как одна группа. Если сервер обнаружит ошибку, то весь пакет не будет выполнен. Разделение команд в пакете осуществляется командой GO. Пакет может быть транзакцией, может быть частью транзакции.

Обозначение однострочного комментария: -- Однострочный комментарий.

Обозначение многострочного комментария:

/* Многострочные

** комментарии*/.

Переменные могут быть двух типов – локальные (начинаются с символа @) и глобальные (начинаются с двух символов @@). Локальные переменные существуют в пределах сеанса, глобальные переменные используются для получения информации о сервере в целом (иногда называют функциями). Локальные переменные объявляются командой DECLARE и значения задаются командами SELECT и SET. Например:

DECLARE @R DATETIME, @C INT

SELECT @C=1
SET @R =getdate()
SELECT @C, @R.

Обычно рекомендуется для присваивания переменных использовать команду SET , а не SELECT, так как первая работает быстрее. В одной команде SELECT можно присваивать значения нескольким переменным. Команда PRINT передает сообщение длиной 1024 байта обработчику сообщений клиента.

Команды условного выполнения:

IF <логическое выражение>
    { <команда> |< блок команд>}
ELSE
    { <команда> | <блок команд>}.

Например, использование команды условного выполнения выглядит так:

IF (SELECT AVG(price) from titles WHERE type = “business”) >$19.6
    PRINT ” Сообщение 1”
ELSE
    PRINT “ Сообщение 2”.

Блок команд – множество команд в операторных скобках BEGIN .. END

Цикл WHILE:

WHILE <логическое условие>
[<команда> |< блок команд>]
[BREAK]
[CONTINUE]
END.

Например, использование цикла с предисловием:

WHILE (SELECT AVG(price) FROM titles) < $25
    BEGIN
        UPDATE titles SET price =price*1.05
        IF (SELECT COUNT(*) FROM titles WHERE price < $15) <10)

```
        CONTINUE
    ELSE
        IF (SELECT MAX(price) FROM titles) >$50
        BREAK
END.
```

Команда WAITFOR переводит запрос в состояние ожидания в течение некоторого интервала или до наступления заданного времени (и потому называется обработчиком события):

WAITFOR {DELAY “время” | TIME “время”}.

Параметр DELAY заставляет пакет или процесс сделать паузу заданной длины (максимум – 24 часа). Параметр TIME останавливает процесс до наступления заданного времени. Время задается в формате hh: mi: ss. Например: WAITFOR TIME “ 22:00:00”.

7.5. Курсоры. Типы курсоров. Работа с курсорами

Курсор – временный набор данных, позволяющий приложениям выполнять действия с отдельными записями, входящими в результат запроса, вместо обработки итогового набора в целом. Курсоры в данном диалекте делятся на три типа: динамические, статические и ключевые.

Динамические курсоры – курсоры, при использовании которых изменения данных отображаются при перемещении курсора. Динамические курсоры используют минимальное количество ресурсов сервера за пределами базы данных. Кроме этого, они в наименьшей степени используют tempdb.

Статические курсоры – курсоры, изолированные от изменений в данных. Сервер сохраняет весь итоговый набор курсора в tempdb при первой выборке.

Ключевые курсоры – курсоры, в которых отображается большая часть изменяемых данных, хотя новые записи и не появляются. При открытии курсора сервер сохраняет ключи возвращаемых записей в tempdb. Новые записи не включаются в курсор, а при обращении к записи, удаленной другим пользователем, происходит ошибка.

Работа с курсором выполняется в следующем порядке:

1. Объявление курсора.
2. Открытие курсора.
3. Выборка записей, модификация или удаление записей из курсора (обработка записей в курсоре).
4. Закрытие курсора.
5. Освобождение ресурсов курсора.

Объявление курсора реализуется следующей командой:

Declare <имя_курсора> [INSENSITIVE] CURSOR [LOCAL | GLOBAL]
[FORWARD ONLY | SCROLL] [STATIC | KEYSET |DYNAMIC]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
FOR <инструкция_Select>
[FOR UPDATE [OF <список столбцов>]].

Объявление курсора с ключевым словом INSENSITIVE заставляет сервер создать копию данных во временной рабочей таблице. При работе с курсором Вы действительно работаете с содержимым временной таблицы, которое не изменяется с модификацией настоящей таблицы. Если при определении курсора не используются опции Read Only и UPDATE, то способности курсора к обновлению определяются следующим:

а) если используется опция ORDER BY в предложении Select или указано INSENSITIVE, курсор будет доступен только для чтения. В противном случае курсор доступен и для обновлений;

б) если в выражении выборки используются GROUP BY, UNION, Distinct или Having, курсоры доступны только для чтения и не будут обновляться изменениями. Результаты передаются во временную таблицу и выбираются оттуда.

Так как курсоры могут считывать строки в переменные, находящиеся в хранимых процедурах или пакетах команд, в инструкции Select нельзя применять маску *.

В курсорах может использоваться любая команда SELECT, удовлетворяющая условиям:

- не может содержать секции INTO, COMPUTE, FOR BROWSE;

- при использовании секций GROUP BY, HAVING, DISTINCT, UNION курсор становится статическим;
- если все таблицы, используемые в курсоре, не имеют уникального индекса, курсор становится статическим;
- если все столбцы, используемые в ORDER BY, не входят в уникальный индекс, динамический курсор преобразуется в ключевой или статический.

Используемые параметры имеют следующий смысл:

Local – курсор локальный по отношению к пакету;

Global – курсор доступен в течение всего подключения;

Forward_only – курсор допускает перебор записей только в прямом направлении, если ключевое слово STATIC | KEYSET |DYNAMIC не указано, то курсор является динамическим;

Scroll – позволяет любые перемещения. Функция, обратная INSENSITIVE. Заставляет курсор читать завершенные изменения и удаления, выполняемые другими процессами сервера. Если вы не объявили курсор с ключевым словом Scroll, единственным методом считывания информации является последовательный проход результирующего набора;

Read Only – эта опция предотвращает любую модификацию данных курсора;

UPDATE – эта опция устанавливается по умолчанию для курсора, определенного в одной таблице.

После объявления курсора для получения возможности считывать данные его следует открыть с помощью команды OPEN <имя курсора>. Открыть можно только объявленный курсор. Когда курсор открыт, SQL распознает все неизвестные переменные и определяет их текущие состояния.

Считывание данных их курсора осуществляется с помощью команды:

FETCH [[NEXT | PRIOR | FIRST | LAST | ABSOLUTE n | @nvar | RELATIVE n | @nvar] FROM] имя_курсора

[INTO @имя_переменной 1, @имя_переменной 2].

Здесь используемые параметры:

NEXT – нормальный порядок считывания (возвращает след. строку);

PRIOR – если курсор определен с ключевым словом Scroll, опция PRIOR позволяет читать предыдущую запись;

FIRST – считывание 1-й записи набора;

LAST – последняя строка набора;

ABSOLUTE n – вызывает считывание n-й строки результирующего набора. Если n – положительное число, то считывание будет с 1-й записи заданное число, если n –отрицательное, то с последней назад заданное число;

Relative n – вызывает считывание n-й записи результирующего набора относительно текущей строки;

Если указываете отрицательное число, то находится нужная строка, отсчитав от текущей назад. Причем, вместо “n” могут использоваться переменные типа int, smallint tinyint;

FROM – следом идет имя курсора из которого считываются данные;

INTO – используется в хранимых процедурах. Когда вы указываете это ключевое слово, данные, возвращаемые курсором, помещаются во временные переменные;

Закрытие курсора осуществляется с помощью команды: Close <имя курсора>.

Курсор следит за вашим местоположением в базе данных и поддерживает активную, открытую структуру указателей на предыдущую и следующую строки информации. При освобождении курсора полностью удаляются любые связанные с ним структуры данных, которые SQL Server держал в открытом состоянии. Освобождение курсора отличается от его закрытия. После освобождения вы больше не сможете открыть данный курсор. Команда освобождения памяти от курсора: Deallocate <имя курсора>.

Хранимая процедура может иметь несколько уровней курсоров, которые используются для получения максимальной гибкой обработки результирующего набора. Для того чтобы получить несколько активных курсоров, их надо открыть. Если курсор возвращает один набор данных, что является наиболее распространенным случаем, большин-

ство языков программирования приложений клиента не различают данные в курсоре и команде select. Используются специальные функции для обращения к данным в курсоре, если он и ассоциированная с ним обработка возвращают несколько результирующих наборов. Курсоры позволяют производить индивидуальные операции с записями в наборе, созданном в команде select. Курсор может включать все таблицы базы данных. Модификация записей с помощью курсора выполняется с помощью команд UPDATE, DELETE через включение секции WHERE CURRENT OF. Модификация разрешается только для однотабличных курсоров.

7.6. Правила, значения по умолчанию, представления

Правило – объект базы данных, который связывается со столбцами таблицы и ограничивает их домены. Правило представляет собой выражение, вычисляемое при вставке или обновлении записи. Если это выражение принимает значение FALSE, команда INSERT или UPDATE, ставшая причиной нарушения, отменяется. Команда создания правила:

CREATE RULE <имя правила>
<@Переменная> <оператор> <выражение>
[{ AND | OR } …].
Например:
CREATE RULE rule1
AS @age between 16 and 21.

Здесь переменная обозначает позицию, в которую подставляется значение столбца. Имя переменной не влияет на выполнение правила. Удаление правила осуществляется командой DROP RULE <имя правила>. Удаляемое правило исчезает из базы данных. Чтобы использовать правило, необходимо ассоциировать его со столбцами. Этот процесс называется связыванием: Sp_bindrule имя правила, ‘имя таблицы. Имя столбца’.

Имена таблиц и столбцов должны заключаться в апострофы, так как аргумент содержит специальный символ – точку.При вызове про-

цедуры устанавливаемое правило не распространяется на данные, уже находящиеся в таблице. После связи правила с таблицей оно начинает действовать автоматически. Если новое правило накладывается поверх предыдущего, оно немедленно вступает в силу. Для разрыва правил со столбцами применяется процедура: Sp_unbindrule имя ‘таблицы.имя столбца’.При вызове этой системной хранимой процедуры не нужно указывать имя правила, так как сервер уже знает, какое правило связано с данным столбцом. Существуют ограничения для правил:

- в правиле могут участвовать только константы, функции и маски редактирования;
- правило не может выполнять просмотр таблицы;
- в правилах не могут сравниваться столбцы таблиц;
- с каждым столбцом может быть связано одно правило;
- если со столбцом связано некоторое правило, оно замещается новым устанавливаемым правилом;
- значения, используемые в правилах, должны быть совместимы с типом данных связанного столбца. При возникновении конфликта снижается быстродействие или возникают ошибки времени выполнения (в зависимости от того, существует ли возможность неявного преобразования типа данных);
- правила не применяются к данным, уже находящимся в таблице, но действуют при обновлении существующих данных;
- правило, связанное со столбцом или пользовательским типом данных, нельзя удалить. Предварительно необходимо разорвать связь правила со всеми столбцами и типами данных;
- рекомендуется использовать ограничения, а не правила.

Значения по умолчанию – объект базы данных, который связывается со столбцами и в базе данных и предоставляет значение столбца в том случае, если столбец отсутствует в команде INSERT. Команда создания значения по умолчанию:

CREATE DEFAULT <имя значения по умолчанию> AS <выражение – константа>.

Например:

CREATE DEFAULT adef AS 18.

Удаление значения по умолчанию: DROP DEFAULT <имя значения по умолчанию>.

Связывание со столбцом: Sp_bindefault <имя значения по умолчанию>, ‘имя таблицы. Имя столбца’. Разрыв связи со столбцом: Sp_unbindefault ‘имя таблицы. Имя столбца’.

Ограничения для значений по умолчанию (з.п.у.):

- в определении значения по умолчанию может использоваться лишь одна константа или функция – логические возможности отсутствуют;
- не могут использоваться механизмы принятия решений или просмотра таблицы, в таких случаях следует использовать триггеры;
- с каждым столбцом может быть связано лишь одно значение по умолчанию;
- при попытке задать второе значение по умолчанию для столбца выдается сообщение по умолчанию;
- тип значения и тип столбца должен совпадать, причем проверка совпадения осуществляется во время выполнения, а не во время создания;
- значения п.у. не применяются к данным, уже находящимся в таблице;
- з.п.у. не должны нарушать никаких правил и ограничений столбца, иначе операции вставки не будут выполнены;
- з.п.у. применяются до применения правила;
- з.п.у. не применяются к данным, уже находящимся в таблице;
- з.п.у. связанное со столбцом или пользовательским типом, нельзя удалить, предварительно связь должна быть разорвана со всеми столбцами.

Пользовательский тип – объект базы данных, используемый для описания доменов столбцов базы данных, является производным от стандартных типов данных. Создание пользовательских типов:

sp_addtype имя пользовательского типа данных, системный тип данных,

Null | No Null,

Просмотр пользовательских типов данных: sp_help. Удаление пользовательского типа: sp_droptype <имя типа>.

Между правилами могут возникнуть конфликты. Для их разрешения должны быть выработаны определенные соглашения:

- правила, связанные со столбцом, имеют приоритет выше, чем правила, которые связаны с типом данных;
- если правило связывается с пользовательским типом данных, оно не замещает правило, связанное со столбцом, характеризуемым этим типом данных.

Представление (Вид) (view) – объект базы данных, но в отличие от таблиц не является физическим хранилищем данных – это логический «вид» физических данных, хранящихся в базе. Его можно представить как хранимое выражение выборки с минимальным набором свойств. Удаление всего представления как объекта никак не повлияет на данные, на основе которых оно построено. В то же время удаление всех записей в представлении может удалить эти записи в исходных таблицах. После определения вида на него можно ссылаться так же, как и на таблицу. Вид не создает постоянную копию выбранных строк и столбцов базы данных. Если в какой-нибудь инструкции вместо имени таблицы указать имя вида, то выполнится инструкция select, входящая в определение вида, и после этого создается временная таблица, которая и отображается на экране. Таким образом, при ссылке на вид выполняется заданная в определении вида инструкция Select. Если вы добавляете столбцы в таблицы, лежащие в основе вида, новые столбцы не появляются в нем до тех пор, пока вы сначала его не удалите и затем снова не определите. Рекомендуется в имени вида добавлять префикс “vw_имя”, иначе объект внешне выглядит как таблица. Виды можно использовать для обеспечения безопасности базы данных. Можно представить такие права доступа к видам, которые будут отличаться от прав доступа к таблицам, лежащим в основе вида. Команда создания вида:

CREAT VIEW <имя вида>

[WITH ENCRYPT]
AS SELECT <инструкция>
[WITH CHECK OPTION].

Здесь опция WITH ENCRYPTION лишает пользователя возможности отображать определение вида в системной таблице Syscomments (то есть никто, кроме создателя вида не может посмотреть его код). При отсутствии этой опции после создания вида его определение сохраняется в системной таблице Syscomments. Для отображения этой информации может использоваться хранимая системная процедура sp_helptext <имя вида>. Просмотреть определение зашифрованного вида невозможно ни одним из способов. Недостатком шифрования определений вида является то, что такие виды не могут воссоздаться при обновлении базы данных или SQL Server. Шифровка используется для безопасности данных, чтобы лишить пользователя возможности просматривать объекты базы данных. Ограничения на использование и определение видов (по поводу команды Select):

1. Нельзя определить вид временной таблицы – это перевалочные структуры базы данных и они существуют только до тех пор, пока данные считываются из постоянных таблиц.

2. Для вида нельзя определить триггер (триггер – объект базы данных, который автоматически выполняется при добавлении, обновлении и удалении строк таблицы.)

3. Нельзя включать в определение вида ORDER BY. Строки в виде неупорядочены. Если бы внутри вида можно было использовать инструкцию Select с предложением ORDER BY, строки получившегося вида стали бы упорядоченными и вид стал бы отличаться от стандартной таблицы базы данных, где данные неупорядочены (хотя секция может быть использована при выборке из вида).

4. В виде нельзя использовать предложение COMPUTE.

5. Внутрь вида нельзя включать предложение Distinct команды Select. Уникальность считываемых с помощью вида строк можно обеспечить, если в таблице, на которую ссылается вид, определить уникальный ключ или индекс.

6. Нельзя использовать внутри вида предложение INTO, т.е. вывод строк не на экран, а в другую таблицу.

Иногда виды делят на простые и составные виды (simple view) определяют для доступа к одной таблице, составные виды (complex view)- доступ к нескольким таблицам, тогда в команде FROM присутствуют несколько таблиц. В некоторых случаях вы можете так изменить данные, что они больше не будут удовлетворять условиям выборки в представлении. Опция WITH CHECK OPTION позволяет контролировать измененные данные, так чтобы критерий выборки не нарушался. Эта опция – работает с командами UPDATE, INSERT и DELETE (запрещает пользователям вставку и обновление записей, не входящих в представление, разрешает модификацию записей, входящих в представление). Для модификации видов используется команда:

ALTER VIEW <имя представления>
[with encryption]
AS < команда select>
[WITH CHECK OPTION].

Команда изменяет существующее представление и не влияет на зависящие от него хранимые процедуры или триггеры, а также не изменяет прав пользователей. Эта команда относится к новым возможностям последней версии SQL Server и позволяет модифицировать представления без их удаления и повторного создания. Представление, используемое в настоящий момент, модифицировать нельзя. Удаление видов осуществляется командой: DROP VIEW <имя_вида>. Удаление вида не влияет на постоянные таблицы, лежащие в их основе, определение вида удаляется из базы данных. Если вы удалите вид, указанный в определении другого вида, то при последующей ссылке на него появится сообщение об ошибке. Можно создать вид, ссылающийся на несколько видов и таблиц. Модификация данных с помощью видов возможна только для простых видов. Вставленные с помощью вида строки можно вставлять даже тогда, когда они не удовлетворяют критерию, заданному в предложении Where внутри определения вида, а вот считать такую строку с помощью этого вида не удастся.

Чтобы не запутаться, к определению вида можно добавить предложение WITH CHECK OPTION, которое запрещает операцию добавления строки посредством вида, если впоследствии с помощью того же вида строку невозможно отобразить. Если при добавлении в вид опре-

деляются не все столбцы, т.е. один или несколько столбцов в виде не заданы, тогда эти столбцы должны допускать значения NULL или Default, иначе строка не будет добавлена. С помощью видов можно удалить строки, даже если в этом виде происходит обращение не ко всем столбцам таблицы. Delete from <имя_вида> WHERE <условие>. Невозможно выполнить операцию удаления строк посредством вида, если в критерии, заданном в предложении WHERE определения вида, не указаны строки, которые вы собираетесь удалять, указывая их в предложении WHERE у инструкции DELETE. Поэтому для запрещения удаления строк, которые не удовлетворяют условию WHERE применять опцию WITH CHECK OPTION не обязательно. Невозможно удалить строку из таблицы посредством вида, если столбец, заданный в предложении WHERE инструкции DELETE, не указан в определении вида. Строки можно удалять непосредственно из таблицы, входящей в определение вида. Любые изменения, сделанные посредством вида, будут выполнены в таблице, указанной в определении вида.

Рекомендуется доступ к базам данных осуществлять через виды (в иерархических и сетевых базах данных данными можно было манипулировать только через виды сущностей, в сетевой модели данных было понятие подсхем – аналог видов). В видах невозможно обновлять столбцы, которые используются в таблицах для создания парных строк, поскольку такие столбцы являются частью разных таблиц. Чтобы обновить столбцы, которые относятся одновременно к двум таблицам, можно с помощью вида обновить значение в одном столбце, а для обновления соответствующего значения в связанном столбце использовать триггер. Определение вида останется в базе данных, даже если лежащая в его основе таблица будет удалена, хотя при обращении к виду будет сообщение об ошибке. Нельзя модифицировать данные в составном виде. Ограничения при создании видов:

- представления могут создаваться только в той базе данных, которую вы используете, хотя к представлениям можно обращаться и из других баз – для этого надо ввести полностью определенное имя представления;
- необходимо указать имя для каждого столбца в представлении, если любой столбец представления является произ-

водным от константы, встроенной функции или математического выражения, если два и более столбца таблицы имеют одинаковые имена, если вы хотите переименовать столбец;

- представление может содержать до 1024 столбцов (раньше – 255);
- при обновлении столбцов записи через представление каждая команда UPDATE должна обновлять лишь одну базовую таблицу (она может ссылаться на несколько таблиц, но должна обновлять только одну).

Представления могут использоваться для упрощения кода, как средство горизонтальной защиты, как средство вертикальной защиты, для маскировки данных, для маскировки изменений в базе данных.

7.7. Хранимые процедуры и функции

Хранимая процедура – это объект базы данных, набор откомпилированных команд – подпрограмма, работающая на сервере. Так как хранимая процедура проходит предварительную компиляцию и оптимизацию, она более эффективна, чем аналогичный набор команд, исполняемый с клиента. Хранимая процедура вызывается по имени, как программный модуль, и может принимать параметры и возвращать результаты. В код хранимой процедуры могут использоваться не только операции извлечения и модификации данных, но и логика ветвления, переменные и другие языковые инструкции, которые делают хранимые процедуры мощным средством реализации логики обработки.

Хранимые процедуры могут быть четырех типов: системные (System), локальные (Local), временные (Temporary) и удаленные (Remote).

Эта классификация по области видимости. Системные процедуры хранятся в базе данных master и начинаются с префикса sp_. Для создания своей системной процедуры необходимо ее создать с именем sp_имя и поместить в master, тогда она будет видима и доступна лю-

бому серверу. Для изменения системной хранимой процедуры нужно сделать ее копию, сохранить оригинальную процедуру под другим именем.

Локальные хранимые процедуры создаются в пользовательских базах данных. Если процедура создается с именем # или ##, то получаете временную хранимую процедуру, помещаемую в базу данных tempdb. Процедуры, начинающиеся с ## – глобальные, они видны любой сессии связи с данным сервером. Процедуры с одним # – локальные и видны только из соединения, в котором они созданы. Когда соединение, в котором создана процедура, закрыто, процедура автоматически уничтожается.

Удаленные процедуры могут вызываться не только с текущего сервера, с которым клиентское приложение установило соединение. При этом текущий сервер является промежуточным звеном. Есть еще один тип хранимых процедур – расширенные хранимые процедуры. Они принципиально отличаются от перечисленных. Это DLL, как правило, написанные на C, и называются хранимыми только потому, что имя хранится в базе данных SQL Server. Сами DLL хранятся как файлы в соответствующем каталоге ОС.

Для создания хранимой процедуры используется команда:

CREATE PROC[EDURE] <имя процедуры> [;< число>]

[@ имя_параметра тип_данных [VARYING] [=<значение>] [OUTPUT] [, …]

[WITH {RECOMPILE | ENCRYPTION}]

[FOR REPLICATION]

AS <инструкции SQL>.

Здесь [; число] – задает номер версии вашей же программы с тем же именем;

@параметр – параметр, передаваемый или получаемый из процедуры. Процедура может иметь до 1024 параметров. Параметры могут принимать значения NULL. Параметр может относиться к любому типу данных, в том числе TEXT, IMAGE, CURSOR. Для параметра типа CURSOR ключевые слова VARYING и OUNPUT являются обязательными;

VARYING – означает, что параметр возвращает итоговый набор, используется только для типа данных CURSOR;

<Значение> – определяет значение параметра по умолчанию. Может быть любой константой или NULL;

OUNPUT – означает, что параметр является выходным, то есть может возвращаться стороне, вызвавшей процедуру. Параметры типа CURSOR обязаны быть выходными;

With RECOMPILE – процедуры будет повторно компилироваться при каждом выполнении;

FOR Replication – опция позволяет выполнить процедуру в процессе репликации.

Нельзя применять опцию WITH RECOMPILE в инструкции Create Procedure, содержащей опцию FOR Replication. Эту опцию применяют для создания процедуры, которая выполняется в режиме репликации.

Для шифрования определения хранимой процедуры, которая записывается в системную таблицу syscomment, в инструкции Create Procedure может быть добавлена опция ENCRYPTION (для запрещения просмотра пользователями информации об определении процедуры). Могут быть изменены только незашифрованные процедуры.

Ограничения: процедуры не могут содержать команд ALTER TABLE, CREATE INDEX, CREATE TABLE, DROP TABLE, DROP INDEX, TRANCATE TABLE, UPDATE STATSTICS.

После создания хранимой процедуры достаточно ввести ее имя в командной строке. Если вызов хранимой процедуры следует не в первой строке, а за выполнением других инструкций, для ее активизации необходимо применять команду

EXEC[UTE] [@код возврата] <имя процедуры> [;<число>]

[@список параметров] [WITH RECOMPILE].

Здесь код возврата – переменная для сохранения кода возврата процедуры,

номер – числовой идентификатор процедуры, при его отсутствии выполняется процедура с максимальным номером версии. Удаление хранимой процедуры:

Drop procedure <имя_хранимой процедуры>.

Процедуры могут встраиваться внутрь других процедур. Возможно до 16 уровней вложенности. Модификация хранимых процедур осуществляется командой:

ALTER PROC[EDURE] имя процедуры [; число]

[@ имя_параметра тип_данных [VARYING] [=значение] [OUTPUT] [, …]

[WITH {RECOMPILE | ENCRYPTION}]

[FOR REPLICATION]

AS ИНСТРУКЦИЯ_SQL

[RETURN [код статуса]].

7.8. Управление триггерами и транзакциями

Триггер – это объект базы данных, являющийся специальным типом хранимой процедуры, которую SQL Server выполняет при операциях в данной таблице. Триггеры выполняют после применения правил, значений по умолчанию и обеспечивают ограничения целостности на уровне строк, обеспечивает ссылочную целостность в базе данных. Выполнение любой команды модификации приводит к срабатыванию триггера.

Для создания триггера нужно быть владельцем базы данных. Триггеры делятся на триггеры вставки, обновления или удаления. Команда создания триггера:

Create Trigger <имя триггера>

ON <имя таблицы>

FOR {insert, UPDATE, Delete}

AS <команды sql>

[RETURN]

При создании триггера можно задать одну, две или три операции. В случае выбора нескольких операций их следует разделить запятыми. SQL Server ограничивает типы инструкций SQL, которые могут быть выполнены при работе с триггером. Большинство ограничений связаны с тем, что в случае возвращения в начальное состояние результатов работы операций обновления, добавления и удаления, вы-

звавших активизацию триггера, результаты выполнения инструкций внутри триггера откатить невозможно. Триггер не может использовать: все команды Create, все инструкции удаления Drop, объектное разрешение доступа: Grand и Revoke, Update Statistics, Reconfigure, операции загрузки Load Database, Load transaction, инструкции физической модификации диска: Disk, временное создание таблиц Select into. Нельзя создавать триггер для вида, а только для базовой таблицы или таблиц. Любая правильная операция Set работает только в период существования триггера. После завершения работы триггера все установки возвращаются в прежнее состояние. Не следует применять операцию select, возвращающую результирующие наборы из триггера, для приложения клиента, требующего специального управления результирующими наборами, независимо от того, делается ли это в хранимой процедуре или нет. Тщательно проверьте, все ли операции select считывают свои значения в локально определенные переменные, доступные в триггере. Удаление триггера: Drop trigger < имя триггера>.

Триггеры могут встраиваться друг в друга. Доступно 16 уровней вложенности. Триггеры становятся вложенными, когда выполнение одного триггера модифицирует таблицу, что вызывает к работе другой триггер. Триггер может вызвать бесконечный цикл. Например таблица A имеет триггер TR_A, который выполняется при обновлении таблицы A. При выполнении триггер A вызывает обновление таблицы B. Эта таблица включает триггер B, который выполняется, когда обновляется таблица B и вызывает обновление таблицы A. Таким образом, если пользователь обновляет любую из этих двух таблиц, два триггера продолжают бесконечно вызывать выполнение друг друга. При возникновении такого состояния SQL Server закрывает или отменяет выполнение триггера. Если триггер вызывает дополнительную модификацию своей базовой таблицы, это не приводит к его рекурсивному выполнению. В этой версии SQL Server нет поддержки повторной (reentrant) или рекурсивной (recursive) хранимой процедуры или триггера.

SQL Server выполняет транзакции автоматически, но системный администратор может управлять ими с помощью опций команды select, установить уровень изолированности пользователей с помощью ко-

манды set и управлять блокировкой для всех пользователей, устанавливая пороговый уровень блокировки.

Определение уровня изолированности пользователей:

SET TRANSACTION ISOLATION Level Read Commited.

Завершенное чтение предполагает разделяемые блокировки на всех переданных в транзакции страницах. При изменении данных устанавливается монопольная блокировка.

Этот уровень изолированности устанавливается по умолчанию. Команда

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED

устанавливает второй уровень изолированности пользователей, уровень незавершенного чтения (то же действие, что и ключевое слово Nolock в отдельной инструкции Select). Никаких разделяемых блокировок, если они имеются в запросе, не выполняется. Если пользователь удаляет целую таблицу, но еще не завершил целиком эту транзакцию, другие пользователи могут из нее читать данные без всяких ошибок, то есть можно экспериментировать с «грязными данными»:

SET TRANSACTION ISOLATION REPEATABLE READ.

Отсутствие повторяемого чтения – это 3-й уровень изолированности пользователей, наиболее эксклюзивный (до завершения транзакции1 транзакция2 изменяет объект А и успешно завершается, транзакция 1 повторно читает объект А и видит его измененное состояние). Этот уровень изолированности гарантирует неизменность читаемых вами данных и невозможность влияния на ваши данные со стороны любой выполняемой другим пользователем транзакции в течение времени вашей жизни. Этот уровень сильно снижает параллельность доступа к данным и используется не часто:

SET TRANSACTION ISOLATION LEVEL SERIALZABLE.

Обеспечивает 4-й уровень изолированности – отсутствие картежей – фантомов (транзакцию 1 выполняет оператор А выборки картежей R с условием выборки S, до завершения транзакции 1, транзакции 2. Вставляет в R запись, удовлетворяющую условию S, тогда при повторении выборки читается новая запись в транзакции 1).

В SQL Server реализован гранулированный метод синхронизационных захватов.

Все блокировки устанавливаются автоматически. Объекты блокировки в SQL Server – база данных, таблица базы данных, включая все данные и индексы, блок страниц – смежная группа из 8 страниц данных или страниц индекса, страница – 8-килобайтная страница данных или страница индекса, ключ – отдельная запись по уникальному индексу, идентификатор записи – отдельная запись по идентификатору записи (по умолчанию страница). По умолчанию любой запрос ведет к блокировке одной полной страницы или расширяет ее до таблицы.

SQL Server дает возможность изменять пороговый уровень блокировок от одной страницы до таблицы. Максимальный порог блокировки – это максимальное число блокированных страниц перед переходом на табличную блокировку, даже если процентный порог расширения блокировки не пройден. Значение порога по умолчанию – 200 страниц. Есть понятие минимального порога блокировки. Табличная блокировка произойдет, только если будут достигнуты минимальный порог и процентный порог расширения. Минимальный порог расширения блокировки предотвращает переход к табличной блокировке для маленьких таблиц, там, где процентный порог расширения достигается часто. Значение по умолчанию – 20 заблокированных страниц. Процентный_порог расширения указывает процент заблокированных страниц, необходимый для перехода на табличную блокировку. Значение по умолчанию – 0, что указывает на возможность табличной блокировки только при достижении максимального порога расширения. Процедура sp_configure позволяет установить 3 типа порогов распространения блокировок. Переход к табличной блокировке называется экскалацией.

Экскалация произойдет при превышении процентного порога блокировки и достижении минимального порога. Действие процентно-

го порога ограничивается минимальным и максимальным порогами. Если процентный порог достигнут, а минимальный нет, экскалация не произойдет. Если процентный порог не достигнут (маленькая таблица), а максимальный превышен (большая таблица), то экскалация произойдет.

Управление блокировками на уровне команды Select, указывая опции оптимизатора Nolock, UpDlock, Tablock, Paglock и Tablockx:

Nolock –отменяет разделяемые блокировки и не принимает во внимание монопольные при исполнении команды, т.е. позволяет читать «грязные данные», аналогична установке уровня Read UNCommitted, но действует только в рамках одной команды.

UPDLOCK – указывает на применение блокировок обновления вместо разделяемых. Позволяет не перекрывать другим процессам доступ на чтение, но к моменту изменения данных позволяет быть уверенным, что модифицироваться будут именно считанные ранее данные.

Tablock – поднимает разделяемую блокировку на уровень таблицы.

В случае UPDATE равнозначно директиве Tablockx.

Tablockx – налагает монопольную блокировку на таблицу.

Paglock – задает страничную блокировку. В сочетании с HoldLock позволяет избежать немедленного наложения табличной блокировки при отсутствии подходящего индекса.

Rowlock – блокировка на уровне записей.

Но при достижении заданного порога экскалации блокировка переводится на уровень таблицы. В команде Select указания оптимизатору устанавливаются следующим образом:

Select <список выбора>
FROM имя таблицы1| вида[(указание оптимизатору)],
имя таблицы n| вида [(указания оптимизатору)].

Создание транзакций реализуется командой:

Begin TRAN [SACTION]
<инструкции TR.SQL>
[ROLLBACK TRAN[SACTION]]
COMMIT TRAN[SACTION].

При появлении команды begin Tran сервер помечает в журнале транзакций базы данных точку возвращения, которая будет использоваться в случае отката транзакций.

Ключевое слово COMMIT TRAN приводит к тому, что все, что было сделано в транзакции, будет реализовано в базе данных.

ROLLBACK Tran – отменяет все инструкции. Нельзя внутри транзакции использовать команды: Create всех видов, DROP всех видов, Alter, Crant и Revoke, Select INTO, TRUNCATE TABLE. Для работы с большими транзакциями используются именованные транзакции и точки сохранения.

7.9. Диагностика и сбор данных. Оптимизация запросов

Оптимизатор запросов определяет наилучший план извлечения данных и наиболее эффективный путь их изменения. Он выбирает способ поиска записей, объединения таблиц и сортировки. Для выбора плана используется система оценок, основанная на стоимости выполнения того или иного варианта.

Стоимость запроса – оценка предполагаемого количества и объема операций ввода/вывода, исходя из предполагаемого количества возвращаемых записей. При этом используется информация о структуре данных, размерах таблиц и индексах, индексная статистика.

Исполнение запроса включает 5 этапов:

1. Синтаксический разбор, состоящий из проверки SQL – предложений и формирования обрабатываемой далее структуры – дерева запроса.

2. Стандартизация – преобразование дерева запроса с целью удаления избыточных структур (например, Between преобразуется в интервал, IN в OR и т.д.).

3. Оптимизация – выработка плана исполнения запроса.

4. Компиляция – перевод выбранного плана в исполняемый код.

5. Исполнение запроса.

Процесс оптимизации запроса включает этапы:

1) анализ запроса, 2) выбор индексов, 3) выбор порядка соединения, 4) выбор наилучшего плана, 5) анализ запроса.

Анализ запроса включает определение аргументов поиска, условий или (OR), условий межтабличного соединения (Join).

Цель определения аргументов поиска – установление возможности использования индексов. Аргументом поиска является условие, позволяющее сузить область поиска данных. К таким относятся условия, задаваемые простыми операторами сравнения *(=, >, <, >=, <=),* где сравниваются поля и константы. Условия поиска можно объединить оператором конъюнкции (AND). Несколько объединенных условий могут быть признаны аргументами поиска, если все поля, участвующие в сравнениях, принадлежат одной таблице.

Не является аргументом поиска условие – A/12>=300, не является аргументом поиска и сравнение двух полей A=B, не являются аргументами поиска и условия с наличием в сравнении отрицания – NOT, NOT IN(), <>; (!=), NOT EXISTS, так как условие отрицания не позволяет ограничить область поиска, а наоборот, требует проверки каждого конкретного значения на предмет удовлетворения условию (может иногда переписать условие, избежав отрицания A<>0. A типа tinyint=> A>0).

Вторая фаза оптимизации – выбор индексов. При этом проверяется, существует ли условие по полю, на котором настроен индекс. При этом условие должно быть аргументом поиска или условием соединения. Если существует индекс, который может быть задействован для доступа к данным, определяется избирательность заданного условия.

Избирательность условия – это отношение количества записей, удовлетворяющих условию, к общему количеству записей в таблице. С помощью индексной статистики, расположенной на статистической странице, оптимизатор определяет, сколько записей удовлетворяет условию. Оптимизатор пытается использовать диаграмму распределения индекса или применяется плотность индекса. Если статистической страницы нет, используются специальные защитные соотношения – «волшебные проценты», которые зависят от условия и имеют величины:

Условие	Избирательность
=	100%

одностороннее сравнение(<,>)	33%
интервал	25%

Статистическая страница не создается, если индекс строится на основе таблицы, в которой нет данных. При выполнении команды TRUNCATE TABLE уничтожаются все статистические страницы для индексов данной таблицы. Чтобы создать статистическую страницу, следует выполнить команду UPDATE STATISTICS. Для каждого индекса оптимизатор строит одну статистическую страницу и включает диаграмму распределения и плотность индекса. Статистическая страница имеет объем 2 Кб. Диаграмма – это гистограмма. Количество шагов равно длине сводного пространства 2016 байт, делится на длину строки шага = 2 байта (под номер шага)+количество байт под значение ключа (тип индекса). Затем число строк делится на количество шагов. В каждый шаг попадает определенное количество значений индексов. Если в запросе задано условие равенства и имеется уникальный индекс, то количество искомых записей по определению не больше 1. При этом оптимизатор полагает количество отбираемых записей единицей и не обращается к статистике независимо от ее наличия. Оценив количество записей в таблице, предположительно удовлетворяющих условию запроса, оптимизатор рассчитывает количество логических чтений страниц, необходимое для их считывания. Число страниц, которые должны быть прочитаны, зависит не только от количества записей, но и от метода доступа, выбранного оптимизатором, для получения этих записей. Оптимизатор производит расчет для всех возможных методов доступа. Количество логических чтений для него равно общему количеству страниц данных в таблице. Если возможно применение кластерного индекса, расчет производится по формуле

Logic_Reads=Cl_IndLevels+N_Data_Pages, где

Количество Кол-во уровней предполагаемое

логических кластерного кол-во страниц

чтений индекса данных = целочисленному кол-ву выбранных записей/ среднее кол-во записей на странице данных.

Среднее кол-во записей на странице во всех формулах определяется по таблице Sysindexes. Если применяется некластерный индекс, модель расчета меняется. SQL Server считывает ключевые значения с

нижнего индексного уровня. Для каждой индексной записи по указателю производится считывание со страниц данных. При этом каждой записи данных соответствует чтение страницы. SQL Server не проверяет, ссылаются ли указатели на одну и ту же страницу данных. Всегда предполагается худший сценарий – все требуемые записи располагаются на разных страницах. Формула для использования некластерного индекса:

Logical_Reads=N_Ind_Levels+N_Ind_Pages+N_Records

N_Ind_Levels – количество уровней некластерного индекса.

N_Ind_Pages – кол-во страниц нижнего индексного уровня = оценочное кол-во выбранных записей /среднее количество записей на странице нижнего индексного уровня/

N_Records – кол-во отобранных записей. Количество чтений страниц положено в основу определения стоимости запроса. При оценке стоимости главным является временной аспект. Чтение рассматривается как наиболее затратный по времени процесс, поэтому остальные процессы оптимизатор игнорирует.

Выбор порядка соединения таблиц – третья основная фаза оптимизации запроса, цель которой – выбор лучшей, с точки зрения минимизации количества операций чтения, последовательности таблиц. Когда в запросе указано несколько таблиц, требуется выбрать оптимальный план соединения таблиц.

Оптимизатор рассматривает возможные варианты, оценивает необходимое количество чтений и выбирает наименее затратный путь. Версия 6.5 использует только одну технику соединения – «вложенные циклы». Многотабличный запрос в принципе обрабатывается следующим образом: таблицы выстраиваются в определенном порядке. Таблица, обрабатываемая первой, называется внешней (Outer), остальные таблицы называются внутренними (inner). Любые соединения таблицы в цепи являются по отношению друг к другу внешней и внутренней. Первую таблицу в цепи часто называют самой внешней (outermost). Выполняется проход на самой внешней таблице. Для каждой записи внешней таблицы из внутренней выбираются записи, удовлетворяющие условиям соединения и другим табличным ограничениям. Про-

цесс продолжается до достижения самой внутренней таблицы – получается последовательность вложенных циклов.

Для оценки стоимости соединения оптимизатор использует избирательность соединения, которая определяет, как много записей внутренней таблицы будет соответствовать одной записи внешней. Если возможна индексная статистика, избирательность соединения = плотности индекса, нет – берется величина, обратная количеству записей в младшей таблице. Для выбора оптимального плана соединения оптимизатор оценивает стоимость всех доступных вариантов обработки индивидуальных таблиц. Далее рассматриваются возможные комбинации последовательности, проведения операции и возможных методов доступа к отдельным таблицам.

7.10. Удаленный доступ к данным

Существует два способа обращения к данным, находящимся на разных серверах: удаленный вызов процедур(RPC) и распределенные запросы. Удаленный вызов процедур (RCP, remote procedure call) используется для доступа к данным, которые находятся на другом SQL Server. Для того чтобы стал возможным RCP, оба сервера необходимо настроить на взаимодействие друг с другом. Для этого имена локального и удаленного сервера должны быть зарегистрированы с помощью хранимой процедуры:

Sp_addserver @server = ‘сервер’

[, @local=’локальный’|NULL], где

@server – имя сервера,

@local – определяет, является ли сервер локальным. Если установлено значение ’локальный’, то сервер считается локальным, если NULL, то удаленным.

Затем выполняется настройка только системным администратором RCP. По умолчанию настройка SQL Server позволяет ему как принимать, так и производить вызовы RCP. Для изменения этих настроек можно воспользоваться процедурой:

Sp_configure ‘remote access’,[1|0]

Reconfigure.

При параметре 1 разрешен доступ к данному серверу, при 0 другие серверы не могут обращаться к этому серверу. Характер взаимодействия можно определить с помощью процедуры:

Sp_serveroption @server = ‘сервер’
[,@optname = ‘наименование параметра’]
[,@optvalue = ‘значение параметра’], где

наименование параметра,	значение параметра
‘rcp’, ‘rcp out’	‘true’, ‘false’.

Если вы устанавливаете для ‘rcp’ значение ‘true’, то локальный сервер сможет принимать вызовы RCP от серверов, определенных в параметре @server. Если то же значение будет установлено для ‘rcp out’, то локальный сервер сможет посылать вызовы RCP на серверы, определенные в аргументе @server. После того как серверы настроены, можно осуществлять удаленное выполнение хранимых процедур, используя их полные имена:

Exec[ute] имя сервера.[база данных].[владелец].имя хранимой процедуры [параметр [,параметр…]].

Удаленная хранимая процедура выполняет любые действия, выполняемые локальными процедурами: возвращать одно число или набор значений, передавать записи и отдельные значения.

Распределенные запросы позволяют обратиться к данным из нескольких источников в одном запросе. Распределенный запрос выполняет на локальном сервере команду SQL, которая влияет на данные удаленного сервера. Перед использованием распределенных запросов подключение должно установить флаги ANSI_NULLS ANSI_WARNINGS: SET ANSI_NULLS ON, SET ANSI_WARNINGS ON.

Связанный сервер представляет собой заранее сконфигурированный источник данных OLE DB. Команды SQL ссылаются на связанный сервер по полному имени объекта. Перед использованием связанного сервера необходимо предварительно настроить локальный сервер на работу с удаленным источником. Полное имя объекта выглядит следующим образом: Имя сервера.имя базы.имя владельца объекта.имя объекта.

Помимо уточнения имени удаленного сервера необходимо указать имя связанного сервера при вызове функции OPENQUERY. Эта функция возвращает набор записей, который может использоваться в

командах SQL вместо таблицы или представления. Функция OPENQUERY предназначена для выполнения сквозных запросов в источниках данных OLE DB: Openquery(связанный сервер,'запрос').

Локальный сервер предварительно должен быть настроен на работу со связанным сервером:

1) сначала связанный сервер надо добавить с помощью процедуры

sp_addlinkedserver @server='сервер'
[,@srvproduct = 'наименование программного продукта']
[,@provider = 'драйвер соединения']
[@datasrc = 'источник данных']
[@location ='местонахождение']
[,@provstr = 'строка драйвера']
[@catalog = 'каталог'].

Здесь:

@server – имя связанного сервера,

@srvproduct – наименование программного продукта, если он указан, то остальные параметры указывать не обязательно,

@provider – уникальный программный идентификатор обработчика OLE DB для источника данных, который должен быть зарегистрирован в реестре,

@datasrc – имя источника данных, известное обработчику OLE DB,

@location – местонахождение базы данных, известное обработчику OLE DB

@provstr – специфическая для провайдера OLE DB строка, которая идентифицирует уникальный источник данных,

@catalog – каталог, используемый при установлении соединения с провайдером OLE DB.

Имеется ещё одна функция, которую можно использовать, если заранее источник не создан:

OPENROWSET('имя провайдера'

{' источник данных' ; ' идентификатор пользователя' ; 'пароль'| 'строка драйвера'}

{[Каталог.] [структура.] объект| 'запрос'}).

При работе с удаленными данными происходят преобразования типа данных и кодировки, используемых на локальном сервере. При выборке данных командами SELECT, UPDATE, INSERT, DELETE удаленные данные преобразуются в тип данных и кодировку локального сервера. При модификации данных данные локального сервера преобразуются к типу данных и кодировке удаленного сервера.

Если кодировка удаленных данных отличается от кодировки локального сервера, результаты запроса могут оказаться бессмысленными. В распределенных запросах разрешаются команды SELECT, состоящие только из секций SELECT, FROM, WHERE.Секция INTO разрешается при условии, что создаваемая таблица находится на локальном сервере. Команды UPDATE, INSERT, DELETE должны соответствовать требованиям OLE DB на удаленном сервере. При работе с удаленной таблицей с помощью курсора секция обновления и удаления WHERE CURRENT OF может выполняться лишь в том случае, если провайдер OLE DB поддерживает данную возможность. Команды полнотекстовой обработки не функционируют в распределенных запросах. Команды CREATE, DROP, ALTER не могут использоваться для удаленных серверов. Курсоры типа STATIC, INSENSITIVE могут ссылаться на удаленные таблицы. Курсоры типа KEYSET могут ссылаться на удаленные таблицы лишь в том случае, если провайдер OLE DB соответствует требованиям, документированным в описании функциональных возможностей курсоров KEYSET. Курсоры FORWARD_ONLY не работают с удаленными таблицами.

Распределенные транзакции работают с данными, находящимися на двух и более серверах. Для явного указания распределенной транзакции используется команда:

Begin distributed transaction [<имя транзакции>].

Глава 8. АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ НА ПРИМЕРЕ SQL SERVER

8.1. Система безопасности. Аутентификация. Учетные записи и роли. Планирование разрешений

Для доступа к данным в SQL Server имеются следующие уровни защиты: *операционная система, SQL Server, база данных, объектов базы данных.* Для доступа к объектам и данным пользователь проходит два уровня проверки безопасности – уровень аутентификации и уровень проверки разрешений.

Аутентификация пользователя – процесс допуска или недопуска к соединению с SQL Server. В SQL Server существует два режима контроля проверки возможности подключения: на уровне с SQL Server с SQL Server (SQL Server authentication) и смешанная система на уровне SQL Server и Windows NT (Windows NT authentication). В первом случае задаются имя пользователя для входа (учетная запись) и пароль во время соединения. Во время установки SQL Server создается только одна учетная запись – sa. Эта учетная запись имеет все права на доступ ко всем объектам всех баз данных. Добавить новую учетную карточку можно с помощью мастера создания нового пользователя SQL Server Security Wizard или New Login через Security дерева административной консоли, либо с помощью системной хранимой процедуры: Sp_addlogin . Удаление имен пользователей для входа:

Sp_droplogin ‘имя пользователя для входа’.

Изменение паролей:

Sp_password ‘старый пароль’,’новый пароль’,’учетная запись пользователя для входа’.

Аутентификация со смешанной системой безопасности предоставляет пользователю использовать имена и пароли, которые исполь-

зуются в сети. Это же можно сделать с помощью системной хранимой процедуры

Sp_grantlogin ‘учетная запись для входа’.

Запретить подключение’:

Sp_denylogin ‘учетная запись для входа’.

Роль – именованный набор прав в рамках сервера или конкретной базы данных. Существуют встроенные роли, пользовательские роли, роли приложений. Встроенные роли, в свою очередь, могут быть серверными или ролями базы данных.

Встроенные серверные роли:

Sysadmin – системный администратор (может выполнять любые действия с SQL Server);

Securityadmin – администратор безопасности (управляет учетными записями сервера);

Serveradmin – администратор сервера (управляет настройками сервера);

Setupadmin – администратор установки (добавляет и удаляет связанные серверы);

Processadmin – администратор процессов (осуществляет управление процессами);

Diskadvin – администратор дисков (осуществляет управление файлами);

Dbcreator – создатель баз данных (создает и изменяет их параметры).

Присвоение встроенной роли можно осуществить с помощью визарда или системной хранимой процедурой: Sp_addsrvrolemember ‘учетная запись пользователя’,’роль’

Отмена роли: Sp_dropsrvrolemember ‘учетная запись пользователя’,’роль’

Просмотр информации о встроенных ролях: Sp_helpsrole, sp_helpsrvrolemember.

Встроенные роли баз данных:

Db_owner – владелец базы данных (может выполнять любые операции с базой данных);

Db_accessadmin – администратор доступа к базе данных (добавляет и удаляет пользователей базы данных);

Db_securityadmin – администратор безопасности базы данных (управляет ролями в базе данных и разрешениями на запуск команд и на работу с объектами базы данных);

Db_ddladmin – администратор DLL (добавляет, изменяет и удаляет объекты базы данных);

Db_backupoperator – администратор резервного копирования (осуществляет резервное копирование базы данных);

Db_datareader – лицо, имеющее право на просмотр базы данных;

Db_datawriter – лицо, имеющее право изменять старые данные и вносить новые;

Db_denydatareader – лицо, которому запрещен просмотр данных;

Db_denydatawriter – лицо, которому запрещено изменение данных.

Для добавления *пользовательских ролей* (добавить новую серверную роль нельзя), а не встроенных, можно воспользоваться системной хранимой процедурой, для которых сделать впоследствии ряд разрешений:

Sp_addrole ‘роль’,’владелец’.

Удаление пользовательской роли:

Sp_droprole ‘роль’.

В каждой базе данных имеется *специальная роль* – роль *public*. Каждый пользователь базы данных – член этой роли. Она не может быть удалена, её нельзя присвоить вручную, ей можно присвоить определенные разрешения, которые затем автоматически передаются всем пользователям базы данных.

Кроме пользовательских ролей можно создавать *роли приложений*. Эти роли нельзя присвоить пользователям. Чтобы использовать предоставленные этой ролью разрешения, её вначале надо активизировать.

Роли приложений создаются хранимой процедурой:

Sp_addapprole ‘роль’,’пароль’.

После этого роль приложения активизируется хранимой процедурой:

Sp_setapprole ‘имя роли’, ‘пароль’.

Применяя роль приложения (иногда говорят прикладную роль), приложение обязано передать пароль при активизации роли (это единственный тип роли, требующий пароль), область видимости прикладной роли – только текущая база данных, прикладную роль, активизированную хранимой процедурой, нельзя отключить в текущей базе данных до отключения пользователя от сети. Удаление текущей роли: Sp_dropaprole. Изменение пароля для прикладной роли в текущей базе данных: sp_approlepassword.

Предоставление разрешений (прав доступа к данным) можно выполнить для следующих объектов: таблиц, значений по умолчанию, правил, индексов, представлениям, триггеров, хранимых процедур. Разрешения представляются учетным записям пользователей пользовательским ролям, ролям приложений. Разрешения бывают объектные и командные. Объектные разрешения управляют доступом к таблицам, видам, хранимым процедурам.

Для каждого объекта могут быть предоставлены различные типы разрешений: Select – таблица, вид (читать), Update– таблица, вид (изменять), INSERT– таблица, вид (добавлять), Delete– таблица, вид (разрешает удалять данные), References– таблица (дает возможность пользователю обращаться к таблице или виду посредством предложения where), EXECUTE–хранимая процедура (разрешает пользователю выполнять хранимую процедуру), ALL – предоставляет любые разрешения.

Типы разрешений Select, UPDATE, INSERT, Delete используются часто. Они определяют возможность пользователя выполнять команды, в опции FROM которых перечислены такие объекты, как таблицы или представления, к которым пользователь должен иметь доступ. Тип разрешения Reference управляет доступом пользователя к поддержке целостности данных. На практике это необходимо только в том случае, когда требуется установить отношение между двумя объектами, имеющими разных владельцев. Тип разрешения EXECUTE позволяет пользователю выполнять хранимые процедуры. Таким образом, пользователь может выполнять какие-либо действия, определяемые хранимыми процедурами, но не имеет доступа непосредственно к

данным таблиц или представлений. Разрешить или запрещать доступ к объектам базы данных может владелец объекта.

Командные разрешения: возможность выполнять следующие команды: Create Database, Create Default, Create Procedure, Create Table, Create View, Backup Database, Backup Log, ALL. Командные разрешения могут представляться системным администратором или владельцем базы данных. Разрешение на создание временных таблиц не требуется. Наиболее простой путь – предоставление или запрещение действий группам пользователей. Рекомендуется следующая последовательность при предоставлении разрешений: предоставьте соответствующие разрешения группе public, предоставьте соответствующие разрешения другим группам, которые находятся в иерархии ниже системной группы public, предоставьте соответствующие разрешения отдельным пользователям.

Предоставление прав осуществляется с помощью команды:

Grant ‘список_прав доступа’ |All
ON [‘таблица’| ‘вид’ [(‘столбец’, …)] | ‘хранимая процедура ’]
TO список_имен ролей или учетных записей | PUBLIC.
Удаление прав осуществляется с помощью команды:
REVOKE список_прав_доступа
ON имя_объекта
TO список_имен.

Здесь Список_прав_доступа – права через запятую или ALL.

Имя_объекта – таблица, вид или хранимая процедура, для которой предоставляется или уничтожается разрешение.

Запрещение доступа к объектам:

DENY { ALL | ‘список разрешений’} TO ‘учетная запись’.

Ограничения на использование разрешений:

- только собственник объекта имеет по умолчанию к нему доступ,
- предоставляя разрешения, можно указать столбцы;

- запрет доступа пользователю, группе или роли имеет приоритет над любым разрешением;
- владелец базы данных не может управлять разрешениями в других базах данных, если он не использует команду SETUSER;
- имена пользователей, имена ролей, паролей должны иметь размер от 1 до 128 символов, не должны содержать обратных слэшей;
- имена пользователей, паролей, ролей не должны совпадать с ключевыми словами и встроенными именами пользователей SQL Server.

8.2. Репликация данных. Типы репликаций

Распределенные данные – это множество копий одних и тех же данных, размещенных на разных серверах. Существует две стратегии распространения данных- распределенные транзакции и репликация. Распределенные транзакции гарантируют согласованность всех копий данных, но отказ транзакции на одном из узлов ведет за собой отказ на всех узлах, поэтому распределенные транзакции используются только при постоянной синхронизации данных. При репликации новые копии дублируются и распространяются из исходной базы данных в другую базу данных, обычно расположенную на другом сервере. На выбор способа распространения данных влияют такие факторы, как задержка, автономность узлов, согласованность и конфликты при обновлении базы данных.

Задержка – это интервал между обновлением распределенных данных. Распределенные транзакции характеризуются практически нулевой задержкой. При репликации возможны задержки от нескольких секунд до нескольких дней.

Автономность узла характеризует степень его независимости. В распределенных транзакциях узлы должны быть постоянно подключены к сети, а в некоторых формах репликаций допускается задержка от нескольких секунд до нескольких дней.

Согласованность предполагает корректное изменение во всех копиях данных. Распределенные транзакции гарантируют полную и постоянную согласованность. Некоторые формы репликаций могут обеспечить согласованность, но с задержкой, а некоторые вообще не гарантируют согласованности модификации данных. При обновлении данных на нескольких узлах могут возникнуть *конфликты*. Распределенные транзакции обеспечивают для нескольких узлов ту же степень согласованности, что и на одном узле. Некоторые формы репликаций предотвращают конфликты, разрешая обновление данных только на одном узле.

Репликация (тиражирование) – это операция пересылки информации из источника в место назначения. Тиражируя, администраторы могут защитить свою информацию или же, наоборот, сделать ее доступной для других. Например: 1)Запросы пользователей выполняются долго. Если администратор знает, что пользователям нужна информация, «возраст» которой не превышает двух часов, чтобы «поместить» данные в этот двухчасовой промежуток, можно воспользоваться средством тиражирования. Создать вторичный сервер, первичный сервер будет ему посылать данные каждые 30 минут, и пользователи будут обращаться к вторичному серверу, не влияя на первичный сервер. После установки и конфигурирования вторичного сервера запросы пользователей выполняются быстрее. 2)Например, есть распределенное финансовое приложение. Самые важные данные хранятся в таблице налогов, которую обслуживает центральный офис. Эта таблица большая и часто изменяется. Администратор должен обслуживать локальные копии этой таблицы на каждом отдельном удаленном узле, его интересуют только ставки налогов его региона. Причем информация всегда должна быть свежей. Для этого используется тиражирование, причем не всех данных, а устанавливается горизонтальное тиражирование с возможностью посылать только нужную информацию на удаленные серверы. 3) Одно из преимуществ тиражирования – это то, что на сервере-подписчике тиражирование таблицы имеет свойство “только для чтения”, позволяющее защитить целостность данных.

Терминология тиражирования.

1. Немедленная гарантированная согласованность (жесткая согласованность) – изменение БД, являющейся источником, немедленно передается в базу – адресат.

2. Отложенная гарантированная согласованность (слабая согласованность) – пересылка данных между узлами осуществляется не в реальное время, хотя передача безошибочная и своевременная. Если компьютеры, содержащие базы данных, соединены постоянно, то изменения данных в одной базе и соответствующие изменения в другой – тиражируемой базе – будет разделять небольшая временная задержка. Если базы данных расположены на компьютерах, которые соединяются периодически, то изменения базы-источника не будут отражаться в базе-адресате, пока не установится соединение.

Основой архитектуры тиражирования со свободным согласованием является журнал транзакций. Изменения, вносимые в журнал транзакций базы-адресата, переносятся на целевую базу данных. Алгоритм планирования тиражирования включает процессы чтения журнала, синхронизации и распределения, которые рассмотрим позже.

3. Издатель (Published) – сервер с исходной базой данных. Издатель у любого подписчика только один. Схема множественного издания не реализуется в SQL Server.

4. Подписчик (Subscriber) – это база данных, которая запрашивает и получает публикации точно так же, как и в случае с обычными журналами. После этого у подписчика имеется копия информации, содержащейся в статьях публикации. Учтите, что подписчик может изменять данные, находящиеся в статьях, но эти изменения не отражаются на базе данных издателя.

Начиная с SQL Server 6.5 можно тиражировать данные в любую базу данных, имеющую драйвер ODBC, а не только в базе данных SQL-сервера.

5. Распространитель (Distributor) – сервер – получатель копии всех изменений в опубликованных данных, сохраняет изменения и открывает доступ к ним соответствующим подписчикам. Реализует распространение специальная системная база данных распространения

(distribution database) и рабочий каталог распространения (distribution working folder).

6. Статья (Article)- это единый реплицируемый элемент данных.

В качестве статьи может выступать:

- таблица (при снимочной репликации может реплицироваться схема таблицы, включая триггеры, а также данные таблицы);
- набор полей таблицы (вертикальная фильтрация);
- набор записей таблицы (в случае горизонтальной фильтрации);
- набор полей и записей таблицы (в случае вертикальной и горизонтальной фильтрации);
- определение хранимой процедуры;
- запуск хранимой процедуры.

7. Публикация – это набор статей, предназначенных для тиражирования. Каждая тема представлена отдельной статьей.

8. Подписка – это конфигурация, определяющая способ получения подписчиком публикации от издателя. Существует два вида подписок: активные и пассивные. Подписка, которая создается и управляется на сервере-издателе, называется пассивной (её создает сервер-издатель, на любую публикацию можно подписать сразу несколько подписчиков). Подписка, созданная на подписчике, называется активной (её создает сервер-подписчик, для использования активных подписок сервер, на котором находится публикация, должен быть постоянно подключен к сети). Активные подписки бывают стандартными (регистрируются на сервере-издателе) и автономными (полностью настраиваются на сервере-подписчике, и информация на издателе не хранится). Последнее идеально подходит для подписчиков, подключающихся через Интернет. Такие подписчики могут подключаться по протоколу FTP.

Для хранения информации тиражирования SQL-сервер использует специальную БД, называемую БД распределения(distribution database)-тиражируемая информация содержится в ней до тех пор, пока не будет передана нужным подписчикам. Пользователи не имеют доступа к этой БД: она предназначена только для работы системы. В

этой базе имеются таблицы 1) Msjob_commands-в каждой транзакции имеется одна или несколько команд. В этой таблице хранятся записи о таких командах (в этой таблице столбцы – номер сервера-издателя, имя опубликованной БД, идентификатор задания(job), идентификатор команды, номер статьи, реальная команда (строка)). 2) Msjob_subscriptions-используется для контроля за тем, какие статьи необходимы конкретным подписчикам (имя подписчика и его номера статей). 3)Msjobs-в этой таблице хранятся реальные транзакции. 4)Mssubscriber_info-подробная информация о заданиях тиражирования, которые должен выполнить SQL Server(тип подписчика: (0-SQL Server; 1-ODBC; идентификатор регистрации; сколько времени должно пройти до момента, когда транзакции будут переданы подписчикам). 5)Mssubscriber_jobs-обеспечивает взаимосвязь между подписчиком и всеми изданиями, которые должны быть выполнены для данного подписчика. 6)Mssubscriber_status-эта таблица контролирует состояние всех изданий, выполняемых для всех подписчиков.

Процесс чтения журнала состоит в том, что SQL-сервер использует журнал транзакций сервера-издателя в качестве источника всей информации тиражирования. За журналом транзакций сервера-издателя следит процесс чтения журнала. Когда этот процесс находит транзакцию, оказывающую воздействие на таблицу, которая была отмечена для тиражирования, он дублирует внесенное изменение в БД распределения. Этот процесс автоматически стартует вместе с запуском SQL-сервера и не требует управления.

Процесс синхронизации: когда в среду тиражирования добавляется новый подписчик, SQL-сервер применяет специальный процесс для синхронизации нового подписчика с существующей конфигурацией тиражирования. Этот процесс обеспечивает корректность системных таблиц и схемы тиражирования. SQL Server автоматически запускает и останавливает данный процесс, как и процесс чтения журнала.

Процесс распределения: для завершения выполнения операции репликации SQL-сервер должен взять информацию в БД распределения и дублировать ее в серверах-подписчиках, для этого используется процесс распределения(distribution). Он стартует автоматически вместе с сервером SQL Server, находит все соответствующие транзакции в БД

распределения и обеспечивает их выполнение в нужном месте назначения.

Чтобы сделать репликацию наиболее эффективной, необходимо предусмотреть следующее:

1. Так как сервер-подписчик может сам тиражировать информацию, то можно построить многоуровневую архитектуру тиражирования. Многоуровневые среды могут работать быстрее, чем одноуровневые, т.к. с помощью промежуточных компьютеров многоуровневая топология может устранить необходимость в дорогостоящих протяженных соединениях.

2. Вне зависимости от частоты выполнения задач тиражирования следует всегда избегать перегрузок сети и системы, которые случаются, когда данные тиражируются на большое число серверов-подписчиков. Рекомендуется выполнять задачи обновления информации не одновременно, а по очереди, когда каждый сервер получает свою информацию в определенное, отличное от других время.

3. После подписки на публикацию нужно использовать автоматическую синхронизацию. Этот метод позволяет объединить задачи синхронизации вместо выполнения каждой из них в отдельности после каждой новой подписки.

4. Роль журнала транзакций усложняется при добавлении в систему функций тиражирования. Нехватка места в журнале транзакций – крайне неприятная ситуация. Когда журнал полон, данные изменяться не могут. Тиражирование, как правило, увеличивает нагрузку на журнал транзакций, т.е. процесс чтения журнала, освобождает транзакции только после их тиражирования, причем между началом транзакций и последующей операцией тиражирования может пройти некоторое время, отсюда следует, что, возможно, журнал транзакций окажется мал или для него нужно предоставить дополнительное пространство.

5. Размещать журнал в быстродействующем устройстве. Независимо от того, применяется тиражирование или нет, журнал транзакций следует всегда располагать в устройстве, использующем быстрый дисковод. Слишком медленный дисковод может оказать негативное влияние на функционирование остальной части системы. Процесс тиражирования выполняет добавочные операции чтения журнала транзакций,

и если до этого дисковод журнала работал на пределе, система еще сильнее перегружается.

6. Избегать длительных транзакций. Длительные транзакции способствуют заполнению журнала, что, в свою очередь, оказывает крайне отрицательное влияние на характеристики системы. В момент, когда транзакция помечается для тиражирования, и момент выполнения тиражирования может разделять значительный промежуток времени, поэтому все, что делается для уменьшения длины транзакций – полезно.

7. Зеркально отображать журнал транзакций.

8. Для того чтобы система могла работать в качестве сервера-издателя или сервера-распределителя, нужны 32 Мбайта ОЗУ, причем минимум 16 Мбайтов д.б. SQL Servery.

9. Не следует тиражировать любую таблицу только потому, что это возможно. Это увеличивает объем работы, выполняемой на всех компьютерах среды, повышает нагрузку на сеть, требует выделения дополнительной дисковой памяти на всех компьютерах подписчиков. Если тиражирование необходимо, то надо решить, как это сделать -по вертикали или горизонтали, что уменьшает поток информации.

10. Если в таблице не определен первичный ключ, SQL-сервер не может реализовать для нее средство тиражирования. Исключение есть в SQL-сервере 6.5-можно выполнить одноразовое тиражирование с помощью операции «моментального снимка»(Snapshot).

11. Рекомендуется отменять проверку ограничения «внешний ключ» добавлением опции Not For Replication в операторы Create Table и Alter Table, т.к. проверка занимает некоторое время.

12. При тиражировании таблиц избегайте дублирования. Это часто происходит, когда администраторы вносят одну и ту же таблицу в разные публикации.

Репликация-дублирование, тиражирование информации между серверами.

Виды реплицируемых данных(виды статей):

1) целые БД(entire database);
2) целые таблицы(entire tables);
3) горизонтальные разделы(horizontal partitions) информации;

4) вертикальные разделы(vertical partitions);

5) выборочные виды(custom views).

1. Если статья – целая БД, то SQL Server осуществляет отслеживание любой активности, связанной с этой БД, и предоставляет информацию об изменениях. Для администратора это самый простой способ передачи информации.

2. При репликации целых таблиц определяется таблица, за которой производится слежение. Любые изменения, сделанные в этой таблице, пересылаются с помощью ядра репликаций(replication engine) в приемный сервер.

3 и 4. При издании горизонтальных разделов таблиц применяется инструкция select, которая выбирает все столбцы информации, но только определенные строки репликации. Вертикальная репликация-выбор только определенных столбцов информации. Возможна комбинация вертикальных и горизонтальных разделов.

5. Можно реплицировать виды.

Если распределение и издание комбинируется на одном сервере, то на нем необходимо иметь 32 Mb ОЗУ.

Столбцы типа text или image не подлежат тиражированию. В тиражировании не могут участвовать следующие объекты: БД model, tempdb, msdb и системные таблицы master, таблицы без первичного ключа, таблицы с типом поля timestamp.

Если инициатива на тиражирование со стороны сервера публикаций – «проталкивание подписки», если со стороны сервера подписки-«вытягивание подписки».

В SQL Server различают следующие виды репликаций: снимочная репликация, транзакционная репликация, репликация сведением (слиянием).

Снимочная репликация (репликация моментальных снимков) – периодическая рассылка всей публикации серверам-подписчикам. Это самая простая в настройке и сопровождении репликация (snapshot). Моментальный снимок создается с помощью агента моментальных списков Snapshot Agent, который записывает структуру и данные. Затем моментальный снимок посылается агентом распределения Distribution Agent, который передает данные подписчикам. Подписчики полу-

чают не изменения к уже имеющимся данным, а полностью новый набор данных, который заменяет прежний. При этом старые данные уничтожаются и на их место записываются новые.

Репликация транзакций – передаются не новые таблицы целиком, а только изменения к опубликованным таблицам и команды на исполнение хранимых процедур с соответствующими параметрами. В процессе репликации участвуют агент моментальных снимков Snapshot Agent, агент распределения Distribution Agent, агент чтения журнала Log Reader Agent. Snapshot Agent используется для предоставления подписчикам начального набора данных. Log Reader Agent просматривает журнал транзакций на предмет наличия записей типа insert, update, delete, относящимся к опубликованным объектам, и осуществляет их пакетное применение к базе данных распределения. Log Reader Agent может работать постоянно или по графику, определенному администратором. Изменения хранятся в базе данных распределения до тех пор, пока Distribution Agent , ответственный за передачу данных подписчикам, не перешлет эти данные. Подписчики могут после этого вносить соответствующие изменения или осуществить перенаправленный вызов процедур в своей базе данных.

Репликация сведением (слиянием) – изменения в базе данных – источнике отслеживаются, а затем синхронизируются с базами данных издателя и подписчика. Snapshot Agent записывает структуру и данные базы данных подписчика и передает их агенту слияния (сведения) Merge Agent, который обеспечивает передачу начальных данных от Snapshot Agent и согласовывает изменения между издателем и подписчиком.

8.3. Перемещение данных

Управление данными является одной из важных задач системного администратора; к процессу управления относятся: передача данных в БД из разных источников; преобразование данных из БД SQL Server в другой формат; распространение данных на другие серверы по сети; загрузка архивной копии данных.

1-й способ перемещения данных: Data Transformation Service (DTS) – служба преобразования данных – используется для импорта, экспорта и трансформации данных между разнородными источниками данных. DTS можно использовать с любыми источниками данных, к которым можно обратиться посредством OLE DB. DTS состоит из четырех компонентов: мастера импорта DTS, мастера экспорта DTS, создателя пакетов DTS и программных интерфейсов DTS. В DTS напрямую не поддерживаются языки C, Visual C++, зато Perl, Java, Visual Basic поддерживаются. DTS переносит данные и их структуру. Другие объекты баз данных (триггеры, правила, хранимые процедуры, значения по умолчанию и т.д.) не переносятся. При использовании мастеров DTS можно указать привязку столбцов и информацию по преобразованию.

Утилита dtswiz – это программа, которая позволяет запустить DTS из командной строки. При её запуске можно указать некоторые параметры для того, чтобы не отвечать на вопросы экранов мастеров импорта и экспорта:

Dtswiz [{/?|{/n|[/u идентификатор учетной записи пользователя для входа] [/p пароль]}

[/f имя файла] {/I | /x} {/r драйвер соединения | [/s имя сервера] [/d база данных] [/y]}}], где

/ f – имя файла экспортируемого или импортируемого;

/ I – говорит об импорте в таблицу базы данных SQL Server;

/x – экспорт из таблицы;

/ r – указывает драйвер соединения;

/ s – указывает на экземпляр SQL Server;

/ u – определяет имя пользователя при соединении;

/ p – указывает пароль;

/ n – определяет, что используется доверенное соединение;

/ d – определяет, какая база данных будет использоваться;

/ y – во время работы утилиты системные базы данных будут спрятаны, их не будет ни в списке баз данных получателя, ни в списке источника.

2-й способ: копирование с помощью команды, определяющей способ *загрузки* данных в таблицу из файла:

Bulk insert ‘имя таблицы’ from имя файла данных [опции].

Главное отличие между программой bcp и командой Bulk insert состоит в том, что последняя может только загружать данные в таблицу, экспортировать же в файл не может.

3-й способ: копирование с помощью программы bcp.

Утилита Bulk Copy Program (bcp.exe или bcp) используется для импорта и экспорта данных из таблиц баз данных SQL Server. Она может работать с файлами самых разных форматов. Когда мы копируем данные из файла, bcp добавляет данные в существующую таблицу, если мы копируем данные в файл, все его предыдущее содержимое будет перезаписано.

При загрузке данных в таблицу эта таблица уже должна быть создана и вы должны иметь разрешения на вставку данных и на осуществление запросов к этой таблице. Номера полей в файле и их количество не обязательно должны совпадать с порядком и количеством столбцов в таблице. Однако, если они не совпадают, вам придется использовать файл для форматирования. Данные в файле должны быть совместимыми с типами данных, определяемыми для столбцов таблицы.

Bcp <имя таблицы> | <имя вида> | ‘запрос’
{in |out | queryout | format}<файл данных>

[-m максимальное количество ошибок][-f файл форматирования] [-e файл ошибок]

[-F первая строка][-L последняя строка][-b размер пакета]

[-n][-c][-w][-N][-6][-q][-C кодировка]

[-t разделитель полей] [-r разделитель строк]

[-I входной файл][-o файл выхода][-a размер пакета]

[-S имя сервера] [-U идентификатор учетной записи пользователя для входа] [-P пароль]

[-T] [-v] [-k] [-E][-h” подсказка [,…n]”].

Здесь имя таблицы – полный путь к ней. Если используется таблица, то указываются параметры: in (из файла в таблицу), out (из таб-

лицы в файл), format (предварительно создается файл форматирования, кроме этого должен быть ключ –f). Если указывается запрос, то должен быть параметр queryout. В запросе должна быть одна команда SELECT или вызов хранимой процедуры, возвращающий один набор данных:

-m – число ошибок, которые могут возникнуть до прекращения работы bcp. Та строка, при операции с которой произошла ошибка, будет проигнорирована. Значение по умолчанию 10;

-f – указывается файл форматирования;

-e – определяет файл ошибок, возникших при выполнении команды;

-F –L используется для определения первой и последней строк. По умолчанию используются все строки в таблице или файле;

-b – определяет количество строк в пакете (по умолчанию пакет равен размеру всего файла);

-n – указывает на необходимость использования родного формата SQL Server;

-c – указывает на необходимость конвертировать все типы данных в текстовый тип. При этом типе хранения считается CHAR, разделитель между полями – символ табуляции, между строк – символ начала строки;

-w – указывает на необходимость конвертировать все типы данных в формат UNICODE;

-N – определяет использовать типы данных SQL для нетекстовых данных. Использование этого параметра в сочетании с –w часто используется для переноса с одного экземпляра SQL на другой;

-6 – используется для SQL 6.5 и более ранних;

-q – указывает, что название таблицы содержит специальные символы, тогда имя таблицы вместе с путем должно быть заключено в двойные кавычки;

-C – определяет страницу кодировки символов для текстовых данных. Этот параметр влияет на коды символов > 127 и < 32;

-t – указывает на разделитель между полями. Разделитель между полями по умолчанию – символ табуляции;

-r – междустроковый разделитель (по умолчанию – символ начала новой строки);

-I –имя входящего файла ответов;

-o –имя исходящего файла, куда будут записаны данные;

-a – размер пакета;

-S –имя экземпляра SQL Server, к которому подключаются для выполнения команды;

-U – имя пользователя (по умолчанию имя локального пользователя);

-P – пароль;

-T – определяет, что будет выполнено доверенное соединение (то есть без указания имени пользователя и пароля);

-v –сообщает номер версии и кому принадлежат права на программу BCP;

-k –указывает, что в полях, не имеющих при копировании значений, будет вставлено NULL (даже если для этих столбцов имеется значение по умолчанию);

-E указывает на то, что в импортируемом файле используется столбец счетчика. Если этот параметр не указан, то эти значения не будут восприняты из файла данных и полям в этом столбце будут присвоены следующие значения. Если это значение используется, то значения столбца будут загружены из файла;

-h – определяет подсказки при выполнении команды bcp.

8.4. Резервное копирование и восстановление баз данных

При возникновении проблем с физическими устройствами, возникновении сбоев в программных средствах, при непреднамеренном уничтожении данных из базы данных придется производить резервное копирование, с последующим восстановлением. Существует два основных способа резервного копирования:

- полное архивирование (или дамп базы данных, содержит все используемые страницы базы данных, full backup) подразумевает создание копий всех страниц с данными в БД, включая все системные

таблицы. Так как среди системных таблиц есть специальная таблица Syslogs , в которую записывается журнал транзакций, при выполнении полного архивирования создается копия и журнала транзакций;

- пошаговое архивирование(incremental backup) предусматривает архивирование только журнала транзакций. Другими словами, создается копия только измененных данных;
- архивирование таблицы(table backup)-позволяет создать архивную копию отдельной таблицы базы данных. Восстановить отдельную таблицу можно и из полного архива БД. Архивирование таблиц следует сочетать с регулярным полномасштабным архивированием данных:

1. Прежде всего при резервном копировании (при архивировании) нужно определить место копии данных или иначе устройство для резервного копирования данных.

2. После того как вы решили, какое устройство вы будете использовать для резервного копирования, необходимо указать это устройство SQL Server, используя хранимую процедуру sp_addumpdevice:

sp_addumpdevice{‘тип устройства’, ’логическое имя’, ’физическое имя’}

[, { тип контроллера|’состояние устройства’}]

тип устройства – определяет тип устройства резервного копирования. Тип данных, используемый для этого аргумента, – varchar(10), значение по умолчанию не используется, и должно быть выбрано одно из следующих значений:

- Disk – жесткий диск
- Pipe – именованный канал
- tape – устройство для записи на магнитную ленту

’логическое имя’ – указывает имя устройства резервного копирования, которое вы затем сможете использовать в командах BUCKUP и RESTORE. Тип данных для этого аргумента – sysname, он не может принимать значение NULL и не имеет значения по умолчанию. Этот аргумент показывает, каким образом вы будете идентифицировать дамп-устройство непосредственно во время резервного копирования

’физическое имя’ – указывает на физическое местонахождение устройства резервного копирования. Физическое имя должно включать

полный путь. Тип данных этого аргумента nvarchar(26), он не может принимать значения NULL и не имеет значения по умолчанию.

Для устройства резервного копирования разрешения должны быть установлены правильно, в противном случае определить и использовать устройства резервного копирования будет невозможно. Также для того, чтобы определить устройство резервного копирования, вы должны быть в базе данных master, так как устройство резервного копирования – это ресурс, определяемый для всего сервера, и записи о нем должны храниться в таблице sysdevices баы данных master.

Например:

Use master

Execsp_addumpdevice disk',’newdumpdev’,’c:\dump\newdump.dat’.

После того как определено устройство резервного копирования, можно создавать резервную копию базы данных. Для этого используется команда BACKUP. Команда создания резервной копии базы данных:

BACKUP DATABASE {база данных|@переменная для базы данных’} to устройство резервного копирования [,…]

[With blocsize – {размер блока|@переменная для размера блока}]

[,description={текст|@переменная для текста}]

[,differential]

[,expiredate={дата|@переменная для даты}|retaindays={количество дней |@переменная для количества дней}]

[,format|noformat]

[,unit|nounit]

[,mediadescription={текст|@переменная для текста}]

[,medianame={имя носителя|@переменная для имени носителя}]

[,name={имя набора резервных копий|@переменная для имени резервных копий}]

[,noskip|skip]

[,{nounload|unload}]

[, restart]

[,stats=проценты].

Команда создания резервной копии журнала транзакций:

BACKUP LOG {база данных|@переменная для базы данных’}

[with no_log|trancate_only}]

to устройство резервного копирования [,…]

[With blocsize – {размер блока|@переменная для размера блока}]

[,description={текст|@переменная для текста}]

[,differential]

[,expiredate={дата|@переменная для даты}|retaindays={количество дней |@переменная для количества дней}]

[,format|noformat]

[,unit|nounit]

[,mediadescription={текст|@переменная для текста}]

[,medianame={имя носителя|@переменная для имени носителя}]

[,name={имя набора резервных копий|@переменная для имени резервных копий}]

[,noskip|skip]

[,{nounload|unload}]

[, restart]

[,stats=проценты], где

<устройство резервного копирования> – имя устройства резервного копирования |disk|tape|pipe;

blocsize – размер физического блока в байтах;

description –используется для добавления описания к набору резервных файлов;

differential –определяет, что будет создана резервная копия только изменений, внесенных в базу данных с момента последнего полного резервного копирования;

expiredate – определяет дату, после которой резервная копия считается устаревшей и может быть перезаписана;

retaindays – определяет сколько дней может пройти до момента, когда резервная копия на устройстве может быть перезаписана;

format|noformat – указывает, что может быть создана новая информация о разметке для всех томов, используемых при копировании;

unit –указывает на то, что резервная копия будет первым файлом на диске или магнитной ленте, но сохраняет существующую информа-

цию о разметке. При этом вся информация на устройстве резервного копирования будет перезаписана;

nounit – параметр по умолчанию, указывает на то, что новая резервная копия будет добавлена к уже существующей информации на носителях устройств резервного копирования;

mediadescription – добавляет описание носителя в текстовом формате;

medianame – указывает имя носителя (до 128 символов) для всего набора носителей, используемого для резервного копирования;

name – определяет имя резервной копии;

noskip – даты устаревания и имена резервных копий должны быть проверены, прежде чем старые будут перезаписаны |skip – отключается проверка дат устаревания и имен;

nounload магнитная лента по умолчанию будет выгружена | unload – будет автоматически перемотана и выгружена;

Restart – проверяет, что если операция резервного копирования будет прервана, ее можно будет перезапустить;

Stats – выводит сообщение после того, как выполнен процент от всего задания (если процент опущен, то через 10 процентов от общего задания выводится сообщение о резервном копировании);

No_log – удаляет неактивную часть журнала транзакций и обрезает файл журнала транзакций. Этот параметр освобождает дисковое пространство. При этом необходимости указывать устройство резервного копирования нет, так как резервная копия журнала транзакций создана не будет. После использования этого параметра записи из журнала транзакций не могут быть восстановлены. Поэтому в целях обеспечения возможности восстановления вы должны немедленно выполнить команду BACKAP DATABASE;

No_Truncate – этот параметр создает резервную копию журнала транзакций без его усечения. Этот параметр используется, если база данных повреждена или помечена как подозрительная (suspect)/.

При выборе стратегии резервного копирования учитывается размер базы данных, то, насколько часто обновляются в ней информация и бюджет, которым вы располагаете. Обычно для баз данных небольшого размера производится периодическое полное резервное копиро-

вание. Для баз данных большого размера полное резервное копирование производится реже, а вместо него выполняются операции по резервному копированию журнала транзакций (который содержит изменения, внесенные с момента последнего резервного копирования). Конкретные параметры зависят от того, сколько времени занимает создание резервной копии базы данных и насколько часто в базу данных заносятся изменения. Резервное копирование журнала транзакций производят, когда необходимо освободить занимаемое им дисковое пространство. Информация о резервном копировании хранится в базах данных master и msdb.

Выбор устройства резервного копирования зависит от нескольких факторов: где вы собираетесь хранить резервные копии, насколько быстро в случае необходимости вам надо будет восстановить данные из резервной копии и будете ли вы куда-либо передавать резервные копии.

Для создания резервной копии базы данных нет необходимости прекращать работу с ней. Кроме того, резервные копии баз данных могут добавляться к существующим наборам резервных копий, так что резервные копии баз данных и журналов транзакций могут находиться на одном физическом устройстве.

Во время выполнения резервного копирования базы данных нельзя выполнять команды ALTER DATABASE с ключами ADD FILE, REMOVE FILE, CREATE INDEX, bcp, BULK INSERT, SELECT INTO. Во время резервного копирования можно делать все транзакции, включая операции без записи в журнал транзакций.

Способ восстановления базы данных зависит от способа создания резервной копии. Если была создана резервная копия только базы данных, то восстанавливаться будет база данных. Если – полная резервная копия, то восстанавливается база данных вместе с журналом транзакции. Для восстановления баз данных и журналов транзакций используется команда RESTORE DATABASE [опции], RESTORE LOG [опции].

Восстановление данных подразумевает загрузку последней архивной копии данных и всех архивных копий журнала транзакций, выполненных с момента архивирования данных.

8.5 Автоматизация решения административных задач. Система оповещений

Система управления базами данных требует регулярного выполнения административных задач, часть из которых можно автоматизировать, например:

- создание индексов заново с новым FILLFACTOR;
- сжатие файлов данных путем устранения пустых страниц в базах данных;
- обновление статистики индексов для более эффективного выполнения запросов;
- проверка целостности данных и страниц данных в базе данных, чтобы убедиться, что данные не повреждены из-за аппаратных или программных сбоев;
- резервное копирование баз данных и журналов транзакций.

Процедуры, которые должны выполняться системным администратором называются *задачами* (jobs). Сотрудники, которые ответственны за выполнение этих задач, называются *операторами* (operators). Сообщения о ситуациях, о которых необходимо известить системного администратора, называются *оповещениями* (alerts). Задачу можно определить один раз и выполнять многократно.

Настройка автоматизированного администрирования включает пять этапов:

1. Сначала необходимо определить себя как оператора.
2. Затем настроить сервер, на котором нет рабочих баз данных, как главный сервер и указать остальные серверы как серверы-получатели.
3. Создать задачу и установить график ее выполнения.
4. Настроить задачу так, чтобы она извещала вас о ходе ее выполнения.
5. После этого запустить службу SQL Server Agent, которая должна работать для выполнения задач.

Для настройки задач можно воспользоваться визардом: Database Maintenance Plan Wizard. С помощью этого средства указываются имя задачи, время выполнения, формулировка задачи или, пользуясь служ-

бой SQL Server Agent, определяются оператор, задача и оповещение. Все эти службы называются диспетчерами. Информацию о необходимости выполнения каких-либо заранее спланированных действий служба SQL Executive получает из данных, записанных в системную БД msdb. На их основании запускается один из диспетчеров, которые и составляют основу SQL DMF. Администратор БД может получать постоянную информацию о состоянии БД путем определения и последующего наблюдения за оповещениями. Например, администратор может получать информацию об остановке сервера или исчерпания свободного пространства БД. С определенным оповещением может быть связана, например, задача расширения пространства, отводимого для размещения БД. Таким образом, создается полностью автоматизированная среда администрирования SQL Server. События часто связываются с ошибками.

ЗАКЛЮЧЕНИЕ

Рамки данного курса «Базы данных» позволили рассмотреть лишь основные теоретические и практические вопросы, связанные с этой интересной областью информационных технологий. Здесь не рассмотрены вопросы реализации систем баз данных, которые больше относятся к курсу «Проектирования баз данных автоматизированных информационных систем», такие как вопросы технологий доступа к данным, системы OLAP, модели хранилищ данных, вопросы параллельных баз данных и многое другие аспекты современных баз данных. Обсуждаемые в данном учебном пособии вопросы являются частью общей информационной культуры и могут быть использованы как стартовая площадка для дальнейшего изучения современных баз данных.

ПРИЛОЖЕНИЕ

Ниже приведены варианты описания предметных областей, которые могут быть использованы в качестве индивидуальных заданий для студентов, проектирующих и реализующих базы данных.

1 Вариант

Создать базу данных «Сведения о жителях города». В базе данных хранятся сведения о жителях города Самары. Житель характеризуется номером паспорта, ФИО, датой рождения, телефонами, полом, категорией (дошкольник, школьник, студент, служащий, пенсионер). Для служащего хранится информация о месте работы, включая код организации, название организации, телефон, должность, зарплата. Хранится информация о жилье, включая почтовый индекс, название района, название улицы, номер дома, номер квартиры. Один житель может иметь в собственности несколько квартир и может работать в нескольких организациях. У одного вида жилья может быть несколько владельцев.

2 Вариант

Создать базу данных «Адресная книга». В базе данных хранятся сведения о владельце адресной книги, включая информацию о номере паспорта, номере полиса, ФИО, адресе, телефонах. Хранится информация о родственниках, включая ФИО, адрес, телефон, степень родства, дату рождения. Хранятся сведения о друзьях и знакомых (ФИО, адрес, телефон, место работы, должность). Все сведения ориентированы на одного человека, у которого друзей и родственников может быть много.

3 Вариант

Создать базу данных «Аптека». В базе данных хранятся сведения о лекарстве, включая название, категорию, название фирмы-производителя, название фирмы-поставщика, форму выпуска, вид упаковки, количество в упаковке, дозировку, срок годности. Хранится информация об аптеке, включая номер аптеки, адрес, телефон, компания,

время начала работы, время окончания работы. Имеются сведения о наличии лекарств в аптеке с указанием количества, цены, даты поступления. В одной аптеке имеется несколько лекарств, одно лекарство может продаваться в разных аптеках, причем одно лекарство может продаваться в разных аптеках по разным ценам.

4 Вариант

Создать базу данных «Студенческая библиотека». В базе данных хранятся сведения о книгах, с указанием ФИО авторов, названия книги, места издания. Хранятся сведения об экземплярах книг (инвентарный номер). Хранятся сведения о читателях, с указанием номера читательского билета, ФИО, номер группы, телефон, адрес читателя. Хранятся сведения о разделах библиотеки (название раздела (учебный или научный абонемент), номер комнаты). В каждом разделе библиотеки имеется много книг, одна и та же книга хранится в одном разделе. Книга имеет несколько экземпляров. Книга может быть выдана только одному читателю, один читатель может получить несколько книг.

5 Вариант

Создать базу данных «Биржа». В базе данных хранятся сведения об акциях предприятий, с указанием наименования предприятия, адреса предприятия, цены акции, количества акций к продаже, величины контрольного пакета акций. Хранятся сведения о покупателях, с указанием ФИО директора или частного лица, адреса, телефона, почтового ящика. Хранятся сведения о сделке, с указанием купленного количества акций, даты сделки. Один и тот же покупатель может участвовать во многих сделках, акции одного предприятия могут быть куплены несколькими покупателями.

6 Вариант

Создать базу данных «Больница». В базе данных хранятся сведения о больных, с указанием номера паспорта, ФИО, адреса, даты рождения, номера полиса, даты поступления, даты выписки, диагноза больного. Хранятся сведения о палатах, с указанием номера палаты, отделения, ФИО лечащего врача. Хранятся сведения о назначениях

пациенту врачом, с указанием названия лекарства, дозировки, количества, периодичности, вида процедуры. Одному пациенту может быть сделано несколько назначений, в одной палате может быть несколько больных, один врач может обслуживать несколько палат.

7 Вариант

Создать базу данных « Бытовое обслуживание населения». В базе данных хранятся сведения о потребителе услуг, с указанием ФИО клиента, адреса, телефона клиента. Хранятся сведения о перечне выполненных услуг, с указанием номера квитанции, вида услуги, описания услуги, единицы измерения, цены за единицу, даты заказа, времени заказа, скидки, даты выполнения, времени выполнения услуги. Хранятся сведения о поставщике услуг, с указанием ФИО поставщика услуг, квалификации. Один поставщик услуг может выполнять несколько видов услуг, один вид услуг может выполняться несколькими поставщиками. Один потребитель услуг может сделать заявки на несколько услуг.

8 Вариант

Создать базу данных «Дума». В базе данных хранятся сведения о депутатах, с указанием ФИО депутата, номера паспорта, даты рождения, округа, должности, оклада депутата. Хранятся сведения об участии депутатов на заседаниях. Хранятся сведения о фракциях, с указанием названия фракции, лидера фракции, телефона, адреса. Хранятся сведения о партии, с указанием названия партии, лидера партии, телефона, адреса. Каждый депутат входит в одну из партий. В одну партию могут входить несколько депутатов. В одну фракцию может входить несколько партий, каждая партия входит в одну фракцию.

9 Вариант

Создать базу данных «Перемещения кадров предприятия». В базе данных хранятся сведения о работниках предприятия, с указанием номера паспорта, ФИО, даты рождения, адреса, домашнего телефона работника. Хранятся сведения об отделах предприятия, с указанием названия отдела, ФИО начальника отдела, телефона,

местоположения на предприятии. Хранятся сведения о назначениях и перемещениях сотрудников предприятия между отделами, с указанием даты, должности, основания, номера приказа. Один работник может трудиться в текущий момент только в одном отделе предприятия, в одном отделе работает много сотрудников, один сотрудник может несколько раз переходить из отдела в отдел с повышением или понижением должности.

10 Вариант

Создать базу данных «Повышение квалификации сотрудников». В базе данных хранятся сведения о сотрудниках предприятия, с указанием номера паспорта, ФИО, даты рождения, адреса, домашнего телефона, рабочего телефона, должности сотрудника. Хранятся сведения о повышении квалификации сотрудника, с указанием номера свидетельства о пройденном повышении квалификации, продолжительности в днях, месте, специальности, годе. Хранятся сведения о результатах повышения квалификации, с указанием номера свидетельства, года переаттестации, специальности, результате. Один сотрудник может неоднократно за время работы на предприятии повышать квалификацию.

11 Вариант

Создать базу данных «Преподаватели кафедры». В базе данных хранятся сведения о преподавателях, с указанием номера паспорта, ФИО, даты рождения, должности, ученого звания, ученой степени. Хранятся сведения о курсах, с указанием названия курса, вида занятий (лекции, практика, лабораторные, курсовой проект), номера семестра, вида отчетности. Хранятся сведения о группах студентов, с указанием номера группы, названия специальности или направления, количества студентов. Преподаватель может читать несколько курсов с разными видами занятий. Один курс с определенным видом занятий в определенном семестре читается одним преподавателем. В одной группе изучается несколько курсов, один курс может читаться в нескольких группах.

12 Вариант

Создать базу данных «Труды кафедры». В базе данных хранятся сведения о преподавателях, с указанием номера паспорта, ФИО, ученого звания, ученой степени. Хранятся сведения о трудах (название труда, место издания, тираж, объем в печатных листах, год издания, цена). Один труд может быть подготовлен несколькими авторами, при этом задается процент участия в изданном труде каждого соавтора. Один преподаватель может подготовить несколько трудов.

13 Вариант

Создать базу данных «Квартплата». В базе данных хранятся сведения о квартиросъемщике, с указанием номера паспорта, ФИО, адреса, площади, количества проживающих, наличия льгот у квартиросъемщика. Хранятся сведения о потреблении, с указанием номера квитанции, года, месяца, вида платежа (газ, электроэнергия, водоснабжение, отопление, горячая вода, канализация), даты оплаты, размера оплаты за потребление. Хранятся сведения о тарифах на одного человека, с указанием года, месяца тарифа и вида платежа. Квартиросъемщик оплачивает квартплату ежемесячно, тарифы могут меняться многократно.

14 Вариант

Создать базу данных «Конференция». В базе данных хранятся сведения об участниках конференции, с указанием номера паспорта, ФИО, даты рождения, организации, адреса, телефона, ученого звания, ученой степени участника. Хранятся сведения о комитетах конференции, с указанием названия комитета, ФИО руководителя, описания помещения, телефона руководителя. Хранятся сведения об оплате участников конференции, с указанием номера квитанции, даты оплаты, суммы, способа оплаты. Участник конференции может входить только в один комитет, в одном комитете присутствует несколько участников. Руководитель комитета также является участником конференции. Участник конференции может оплачивать частями несколько раз. В одной квитанции об оплате может быть оплачено участие в конференции нескольких участников.

15 Вариант

Создать базу данных «Продажа недвижимости». В базе данных хранятся сведения о квартирах, с указанием района, названия улицы, номера дома, номера квартиры, типа дома, этажа, общей площади, жилой площади, коэффициенте комфортности квартиры. Хранятся сведения о покупателях, с указанием номера паспорта, ФИО, телефона, места работы, должности. Хранятся сведения о продавцах, с указанием номера паспорта, ФИО, телефона продавца. Хранятся сведения о сделках, с указанием номера договора, даты, стоимости продажи. Один покупатель может совершить покупку нескольких квартир, один продавец может продавать жилье неоднократно.

16 Вариант

Создать базу данных «Лекарства». В базе данных хранятся сведения о лекарстве, с указанием названия, категории лекарства, формы выпуска, упаковки, количества в упаковке, дозировки, срока годности, цены лекарства. Хранятся сведения о производителях лекарственных препаратов, с указанием названия фирмы, адреса, телефона, ФИО представителя фирмы. Хранятся сведения о поставщиках в аптеки лекарства, с указанием названия фирмы поставщика, адреса, телефона, ФИО поставщика. Хранятся сведения об аптеках с указанием названия аптеки, адреса и телефона. Один производитель выпускает много лекарственных препаратов. Одно лекарство может быть выпущено только одним производителем, иначе изменится его название. Поставщик может поставить несколько лекарственных препаратов. Одно лекарство может поставляться в аптеку несколькими поставщиками, при этом факт поставки уточняется количеством, датой поступления и ценой.

17 Вариант

Создать базу данных «Нагрузка кафедры». В базе данных хранятся сведения о преподавателях кафедры, с указанием ФИО преподавателя, должности, общей плановой нагрузки в часах преподавателя. Хранятся сведения о нагрузке преподавателя по предмету, с указанием названия предмета, количества студентов, вида отчетности (зачет, экзамен), объема лекционных часов, лабораторных часов, практических, курсовых часов. Хранится информация о нагрузке кафедры, с указанием названия предмета, количества студентов, объема лекционных ча-

сов, лабораторных, практических, курсовых. На кафедре работает несколько преподавателей, один преподаватель ведет несколько предметов, один предмет может читаться несколькими преподавателями с учетом их должности.

18 Вариант

Создать базу данных «Начисление зарплаты». В базе данных хранятся сведения о работниках, предприятии, с указание ФИО сотрудника, номера паспорта, разряда, должности, наличия льгот при налогообложении. Хранится информация о штатном расписании на предприятии, с указанием названия должности, количества вакансий, количества занятых сотрудников на должности, оклада. Хранится информация о налогах, с указанием вида налога, процентной ставки от оклада на налог. Хранится информация о надбавках, с указанием названия надбавки и процентной ставки от оклада на надбавку. Хранится информация о зарплате сотрудников, с указанием даты зарплаты, всех налогов и надбавок, а также общей заработной платы в указанную дату. Каждый работник получает зарплату с учетом оклада по штатному расписанию, надбавок, налогов ежемесячно.

19 Вариант

Создать базу данных «Поликлиника». В базе данных хранятся сведения о врачах, с указанием ФИО врача, даты рождения врача, специальности, стажа по специальности. Хранятся сведения о днях приема врачей, с указанием дня недели, времени начала и окончания приёма. Хранятся сведения о больных, с указанием ФИО посетителя, номера полиса, адреса, пола, даты рождения. Хранятся сведения о посещениях пациентов врачей, с указанием даты, времени, диагноза. Один врач может принять нескольких пациентов, один пациент посетить несколько врачей в разное время или дни.

20 Вариант

Создать базу данных «Путевки». В базе данных хранится информация о путевках, с указанием кода путевки, места отдыха, начала, окончания, количества, стоимости. Хранятся сведения о заявлениях на путевки, с указанием ФИО написавшего заявление, о дате подачи, дате

рассмотрения, месте работы, должности, стаже, результате рассмотрения. На одну путевку может быть несколько заявлений. В заявлении указывается только одна путевка. Один и тот же сотрудник может получать путевки многократно в разные даты.

21 Вариант

Создать базу данных «Расписание». В базе данных хранятся сведения о преподавателях, с указанием ФИО, должности. Хранятся сведения о парах, с указанием номера пары, начало1, окончание1, начало2, окончание2. Хранятся сведения о записи в расписании, с указанием недели (первая или вторая), дня недели, номера пары, группы, вида занятий, аудитории. Хранятся сведения о предметах, с указанием названия предмета. В расписании учитываются предмет и преподаватель, его ведущий. Преподаватель может вести занятия в разные дни, в один день несколько преподавателей читают разные курсы.

22 Вариант

Создать базу данных «Рецепты приготовления блюд». В базе данных хранятся сведения о рецептах, с указанием названия рецепта, описания, типа пищи, времени приготовления, количества порций, калорийности. Хранятся сведения о компонентах рецепта, с указанием названия компонента, количества, цены, единицы измерения, калорийности. Один компонент может входить в несколько рецептов, в одном рецепте может быть несколько компонент.

23 Вариант

Создать базу данных «Сведения о сессии». В базе данных хранятся сведения о студентах, с указанием номера зачетки, ФИО студента, номера группы. Хранятся сведения о преподавателях, с указанием ФИО преподавателя, названия кафедры, должности. Хранятся сведения о предметах, сдаваемых в данную сессию, с указанием названия предмета, вида отчетности (экзамен, зачет, зачет с оценкой). Хранятся сведения о результатах сдачи студентов группы конкретного предмета конкретному преподавателю. Студенты сдают в сессию несколько предметов, один предмет сдают несколько студентов группы, препода-

ватель может принимать в сессию несколько предметов, один предмет в данную сессию принимает один преподаватель.

24 Вариант

Создать базу данных «Склад». В базе данных хранятся сведения о поставщиках товаров на склад, с указанием номера поставщика, банковских реквизитов, адреса, телефона. Хранятся сведения о получателях товаров со склада, с указанием номера получателя, банковских реквизитов, адреса, телефона. Хранятся сведения о факте прихода на склад товара, с указанием номера накладной прихода, наименования товара, описания товара, ед. измерения, количества прихода, цены, даты поступления. Хранится информация о факте расхода товара со склада, с указанием номера накладной расхода, наименования товара, о количестве расхода, дате расхода. Хранятся сведения о самом товаре на складе, с указанием его наименования и количества на складе. Поставщик может поставлять несколько видов товаров, товар могут получать несколько получателей.

25 Вариант

Создать базу данных «Спорт». В базе данных хранятся сведения о сотрудниках спортивного комплекса, с указанием ФИО сотрудника, адреса, телефона, даты рождения, должности, стажа работы, образования, оклада. Хранятся сведения о клиентах, с указанием ФИО клиента, адреса, даты рождения, пола, места работы, количества занятий, телефона. Хранятся сведения о занятиях, с указанием вида занятий, даты начала занятий, количества занятий, цены. Хранятся сведения о спорткомплексе, с указанием названия спорткомплекса, адреса, телефона, ФИО директора. Один клиент может заниматься разными видами спорта, причем занятия одного вида могут проводиться различными сотрудниками.

26 Вариант

Создать базу данных « Телефонные переговоры». В базе данных хранятся сведения об абонентах, с указанием номера телефона абонента, ФИО, адреса, льготы. Хранятся сведения о переговорах абонентов,

с указанием кода города, даты разговора, времени разговора, о количестве минут разговора, дате оплаты за разговор. Хранятся сведения о кодах города и тарифах города, с указанием кода города, названия города, тарифа за одну минуту (цена минуты). Тариф за одну минуту меняется в зависимости от времени суток разговора, поэтому хранятся сведения о тарифных коэффициентах, включая указание начала периода, конца периода, значение коэффициента. Один абонент может звонить неоднократно. Стоимость переговоров учитывает город и период времени звонка.

27 Вариант

Создать базу данных «Товары». В базе данных хранятся сведения о товарах городского склада, с указанием кода товара, наименования товара, цены, текущего количества, диапазона скидок, даты поступления товара на склад. Хранятся сведения о магазинах, где имеется товар, с указанием наименования магазина, адреса, телефонов, ФИО директора магазина. Хранится информация о фирмах, поставляющих товар в магазины, с указанием наименования, адреса, телефона, ФИО директора. Фирма может поставлять несколько товаров в различные магазины.

28 Вариант

Создать базу данных «Фототека». В базе данных хранятся сведения о пленках, с указанием кода пленки, цены, чувствительности, типа пленки (цветная, негативная), количества кадров, даты начала съемки, даты проявления, места проявления, места хранения. Хранится информация о кадрах, с указанием даты съёмки, места съемки, темы, ФИО участников. Хранятся сведения о фотографиях, с указанием размера, бумаги, количества, ФИО изготовителя, цены, места нахождения. В фототеке ведется учет изготовленных фотографий с учетом пленки и кадра. Из одной пленки может быть напечатано несколько кадров и для каждого кадра несколько фотографий.

29 Вариант

Создать базу данных «Футбольный турнир». В базе данных хранятся сведения о команде, с указанием названия команды, ФИО главного тренера, ФИО продюсера. Хранятся сведения об игроках команд, с указанием ФИО, номера, амплуа, возраста. Хранятся сведения о матчах, с указанием номера матча, название команды1, название команды2, даты матча, места, времени, результата, количества зрителей, средней цены билета. Хранятся сведения о составе игроков каждой из двух команд на игру, с указанием амплуа во время игры и результативности игроков. В одной игре могут играть разные игроки, то есть состав игроков команды на игру может меняться. Одна команда может участвовать в нескольких матчах, один игрок может входить в одну команду, в одной команде несколько игроков.

30 Вариант

Создать базу данных «Личное имущество». В базе данных хранятся сведения об имуществе, с указанием названия, описания, стоимости, даты приобретения, места нахождения, принадлежности. Хранятся сведения о ремонте имущества с указанием вида ремонта, описания ремонта, стоимости ремонта, организации, выполнившей ремонт, ФИО мастера, даты ремонта. Хранятся сведения о категориях имущества (номер, название, описание). Категория имущества включает несколько конкретных наименований имущества, имущество может неоднократно подвергаться ремонту.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт. ; пер. с англ. – 6-е изд: – Киев; М.; СПб.: Издательский дом «Вильямс»,1999. – 848 с.
2. Конноли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Конноли, К.Бегг ; пер. с англ. – 3-е изд. – М.: Издательский дом «Вильямс», 2003.-1440 с.
3. Хомоненко, А.Д. Базы данных: учеб. для высш. учеб. заведений / А.Д. Хомоненко, В.М. Цыганков, М.Г. Мальцев ; под ред. проф. А.Д. Хомоненко. – СПб.: КОРОНА принт, 2000. – 416 с.
4. Хансен, Г. Базы данных: разработка и управление / Г. Хансен, Д. Хансен ; пер. с англ. – М.: ЗАО «Издательство БИНОМ», 1999. – 704 с.
5. Клайн, К. SQL: справочник / К. Клайн ; пер. с англ. – 3-е изд. – М.: КУДИЦ-ОБРАЗ, 2010 – 832 с.
6. Маклаков, С.В. Создание информационных систем с AllFusion Modeling Suite / С.В. Маклаков. – М.: ДИАЛОГ-МИФИ, 2003. – 432 с.
7. Гарбуз, Дж. Database Design on SQL Server 7. Сертификационный экзамен – экстерном (экзамен 70-029) / Дж. Гарбуз, Д. Паскузи, Э. Чанг. – СПб.: Издательство «Питер», 2000. – 560 с.
8. Гарбуз, Дж. Administering SQL Server 7. Сертификационный экзамен – экстерном (экзамен 70-028) / Дж. Гарбуз, Д. Паскузи, Э. Чанг. – СПб.: Издательство «Питер», 2000. – 560 с.

Учебное издание

Чигарина Елена Ивановна

БАЗЫ ДАННЫХ

Учебное пособие

Редактор Т.К. К р е т и н и н а
Доверстка Л.Р. Д м и т р и е н к о

Подписано в печать 22.05.2015. Формат 60х84 1/16.
Бумага офсетная. Печать офсетная.
Печ. л. 13,0.
Тираж 100 экз. Заказ . Арт. 17/2015.

федеральное государственное автономное образовательное учреждение
высшего образования «Самарский государственный аэрокосмический
университет имени академика С.П.Королева
(национальный исследовательский университет)»
443086 Самара, Московское шоссе, 34.

Изд-во СГАУ. 443086 Самара, Московское шоссе, 34.