П.В. Сенченко

ОРГАНИЗАЦИЯ БАЗ ДАННЫХ

Учебное пособие

Министерство образования Российской Федерации
Томский государственный университет систем управления
и радиоэлектроники (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

П.В. Сенченко

ОРГАНИЗАЦИЯ БАЗ ДАННЫХ

Учебное пособие

для студентов специальностей
220200 — «Автоматизированные системы обработки информации и управления»
061000 — «Государственное и муниципальное управление»

Томск 2004

Сенченко П.В.

Организация баз данных: Учебное пособие. — Томск: Томск. гос. ун-т систем управления и радиоэлектроники, 2004. — 170 с.

В учебном пособии рассматриваются вопросы организации баз данных. Изложены принципы проектирования реляционных баз данных, нормализации отношений. Подробно рассматриваются операции реляционной алгебры, синтаксис и применение языка SQL. Даются характеристики баз данных различных поколений. Материал подготовлен на основе учебного курса, который читается автором в Томском государственном университете систем управления и радиоэлектроники.

Учебное пособие ориентировано на студентов специальностей 220200 — «Автоматизированные системы обработки информации управления» и 061000 — «Государственное и муниципальное управление», а также студентов родственных специальностей и разработчиков баз данных.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Учебное пособие составлено в соответствии с требованиями Государственного образовательного стандарта по дисциплине «Базы данных» специальности 220200 – «Автоматизированные системы обработки информации и управления», а также в соответствии с программой курса «Организация баз данных» для специальности 061000 – «Государственное и муниципальное управление».

Знание материала, изложенного в данном пособии должно не только помочь в освоении новых технологий квалифицированным программистом, но и дать общий базис знаний по организации баз данных (БД) пользователям компьютеров.

Целью дисциплины «Базы данных» является обучение способам организации и методам проектирования баз данных, технологии их использования в системах обработки информации и управления.

В данном учебном пособии предлагается к изучению материал, позволяющий студенту получить знания по основным разделам дисциплины:

– положения концепции баз данных, теория структуризации данных, принципы построения баз данных и методы доступа к ним;

– современные системы управления базами данных и их место в системах обработки информации;

– новейшие методики проектирования баз данных.

В результате изучения дисциплины студент приобретает знания в области:

– построения концептуальной информационной модели предметной области;

– реализации простых информационных технологий в экранном интерфейсе современных систем управления базами данных.

Знание организации БД позволит на профессиональном уровне работать с конкретными прикладными системами по ведению и обработке информации.

1. БАЗЫ ДАННЫХ И ФАЙЛОВЫЕ СИСТЕМЫ. ОБОСНОВАНИЕ КОНЦЕПЦИИ БД

1.1. Направления развития вычислительной техники

С самого начала развития вычислительной техники образовались два основных направления ее использования. Первое направление (середина 50-х годов) характеризовалось широкомасштабным применением электронно-вычислительной техники для выполнения математических расчетов, которые тяжело и долго или вообще невозможно производить вручную. Становление этого направления способствовало интенсификации методов численного решения сложных математических задач; развитию класса языков программирования, предназначенных для записи в программном коде численных алгоритмов; возникновению обратной связи с разработчиками новых архитектур ЭВМ. При этом объем исходных данных был соизмерим с объемом оперативной памяти, хотя и был меньше. Одним из недостатков первого направления являлась невозможность повторного использования исходных данных.

Второе направление (первый этап развития — 60-е годы), которое непосредственно касается темы нашего курса, — это использование средств вычислительной техники в автоматизированных информационных системах. Информационная система представляет собой программный комплекс, функции которого состоят в обеспечении надежного хранения информации в памяти компьютера, выполнении операций по обработке информации для данного приложения, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация может иметь сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах, системы складского учета и т.д.[1].

Естественно, что второе направление возникло несколько позже первого. Это связано с тем, что на начальном этапе развития вычислительной техники компьютеры обладали ограниченными возможностями в части памяти. Говорить о надежном и долговременном хранении информации можно только при наличии энергонезависимых запоминающих устройств, т.е. сохраняющих информацию после выключения электрического питания. Оперативная память этим свойством обычно не обладает. В начале создания средств вычислительной техники ис-

пользовались несколько типов носителей информации во внешней памяти: магнитные ленты, перфоленты, перфокарты и барабаны. При этом перфоленты и перфокарты могли обеспечить только хранение информации без возможности ее перезаписи, емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали лишь последовательный доступ к данным. Магнитные же барабаны, отдаленно напоминающие современные магнитные диски давали возможность произвольного доступа к данным, но были ограниченного размера.

Легко заметить, что указанные ограничения не очень существенны для чисто численных расчетов. Даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти таким образом, чтобы программа работала как можно быстрее и сохраняла данные после окончательных вычислений на любом из вышеперечисленных носителей. Однако для информационных систем, в которых потребность в текущих данных определяется пользователем, наличие только магнитных лент и барабанов и тем более перфокарт и перфолент недостаточно. Одним из естественных требований к таким системам является средняя скорость выполнения операций.

Требования к вычислительной технике со стороны нечисленных приложений вызвали появление съемных магнитных дисков с подвижными (плавающими) головками, что явилось революцией в истории вычислительной техники. Эти устройства внешней памяти обладали существенно большей емкостью, чем магнитные барабаны, обеспечивали удовлетворительную скорость доступа к данным в режиме произвольной выборки, а возможность смены дискового пакета на устройстве позволяла иметь практически неограниченный архив данных, перфокарты и перфоленты ушли в историю.

С появлением магнитных дисков началось развитие систем управления данными во внешней памяти. До этого каждая прикладная программа, которой требовалось сохранение данных во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной и внешней памятью с помощью программно-аппаратных средств низкого уровня, машинных команд и вызовов соответствующих программ операционной системы. Такой режим работы затруднял поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной программе при-

ходилось решать проблемы именования частей данных и структуризации данных во внешней памяти [1].

1.2. Файл и области применения файлов

Историческим шагом в развитии информационных систем явился переход к использованию централизованных систем управления файлами. С точки зрения прикладной программы ***файл — это именованная область внешней памяти***, в которую можно записывать и из которой можно считывать данные. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами и, возможно, от типа файла. Система управления файлами берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным [1].

Область применения файлов достаточно обширна, прежде всего, файлы применяются для хранения различной текстовой информации: документов, текстов программ, электронных таблиц и т.д. Такие файлы обычно образуются и модифицируются с помощью различных текстовых редакторов или текстовых процессоров, наиболее распространенным из которых является MS Word. Структура текстовых файлов представляется в виде либо последовательности записей, содержащих строки текста, либо последовательности байтов, среди которых встречаются специальные служебные символы.

Файлы, содержащие тексты программ, используются как входные тексты компиляторов и интерпретаторов языков программирования, которые, в свою очередь, формируют файлы, содержащие объектные модули и подпрограммы. Исходя из [1], с точки зрения файловой системы объектные файлы и библиотеки дополнительных функций файловой системы также обладают очень простой структурой, представляющей собой последовательность записей или байтов. Система программирования накладывает на эту структуру более сложную и специфичную для этой системы структуру объектного модуля. Логическая структура объектного модуля неизвестна файловой системе, эта структура поддерживается средствами конкретного языка программирования.

Файловые системы в большинстве своем обычно обеспечивают формирование и обработку слабоструктурированной информации, оставляя дальнейшую структуризацию непосредственно прикладным программам.

1.3. Понятие СУБД и информационных систем

При разработке информационной системы (ИС) определенной предметной области (*под предметной областью будем понимать любой тип организационной структуры, например, банк, университет, завод, больница*) основным фактором является необходимость обеспечения возможности надежного хранения и обработки постоянно существующей информации, при этом необходимо обеспечить выполнение дополнительных функций, не всегда свойственных простым файловым системам. Структура информации в ИС может быть очень сложна для интерпретации, и хотя структуры данных различны в разных информационных системах, между ними часто бывает много общего. На первом этапе использования вычислительной техники для обработки информации проблемы структуризации данных решались индивидуально в каждой информационной системе.

Следует отметить, что в начале 60-х годов преобладал так называемый позадачный подход в использовании исходной информации, который характеризуется тем, что для каждой программы обработки используются «свои» файлы исходных данных.

Основной недостаток позадачного подхода — дублирование исходных данных в различных файлах, что приводит к серьезным проблемам при их обновлении [2].

Несомненно, что при наличии нескольких программ, оперирующих с файлами, относящимися к одной предметной области, возникают проблемы с поддержкой актуализации хранимых данных в приемлемое для конечных пользователей время.

С течением времени создавались необходимые надстройки над файловыми системами (библиотеки программ), подобно тому, как это делается в компиляторах, редакторах и т.д. Но поскольку информационные системы требуют поддержки сложных структур данных, эти индивидуальные дополнительные средства управления данными, являющиеся существенной частью ИС, в основном дублировались в различных системах. ***Стремление выделить и обобщить составную часть информационных систем, ответственную за управление сложно структурированными данными, явилось первой побудительной причиной создания СУБД.***

Однако изначально невозможно было обойтись только общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения и способы доступа к

данным. Покажем это на примере, за основу возьмем пример предложенный в [1].

Пусть, необходимо реализовать простую информационную систему, в которой ведется учет студентов ВУЗа. Система должна выполнять следующие действия: выдавать перечень студентов по группам, поддерживать возможность перевода студента из одной группы в другую, приема абитуриентов и возможность фиксирования отчисления студентов. При этом для каждой группы отдела должна поддерживаться возможность получения информации о кураторе группы, общего числа студентов, и среднем размере стипендии студентов группы за последний месяц и т.д. Кроме того, для каждого студента должна поддерживаться возможность выдачи номера по ФИО студента, выдачи полного имени по номеру студенческого билета.

Разработать такую информационную систему можно с использованием простой файловой системы и пользоваться при этом одним файлом, расширив базовые возможности программной файловой системы за счет специальной библиотеки функций. Поскольку минимальной информационной единицей в примере является студент, в этом файле содержится одна запись для каждого студента. Запись должна содержать следующие поля: ФИО студента (ФИО_СТУД); номер студенческого билета (НОМЕР_СТУД_БИЛЕТА); размер стипендии (СТИП);); средний размер стипендии (СРЕД_СТИП); количество студентов в группе (СТУД_КОЛ); номер группы (НОМЕР_ГРУППЫ). Используя в системе один файл для хранения информации, необходимо учесть, что эта же запись должна содержать ФИО Куратора (КУРАТОР).

Заданные в условии требования предполагают, что в такой информационной системе должен обеспечиваться доступ к этому файлу по уникальным ключам, недублируемым в разных записях: ФИО_СТУД и НОМЕР_СТУД_ БИЛЕТА. Кроме того, должна обеспечиваться возможность выбора всех записей с общем значением НОМЕР_ГРУППЫ, то есть доступ по неуникальному ключу. Для того чтобы получить общее число студентов группы или средний размер стипендии, каждый раз при выполнении такой функции система должна будет выбрать все записи о студентах группы и посчитать соответствующие общие значения.

Реализация такой системы на базе файловой системы, требует не только создания достаточно сложной надстройки для организации доступа к файлам, но и вызывает требование существенной избыточности хранения (для каждого студента группы повторяется ФИО Куратора) и выполнения массовой выборки и вычислений для получения информации о средней стипендии. Кроме того, если в ходе эксплуатации

системы возникнет необходимость, например, выдавать списки студентов, не получающих стипендию, то придется либо полностью просматривать файл, либо реструктурировать его, объявляя ключевым поле СТИП.

Одним из способов структуризации хранения информации в такой системе является возможность поддерживать два файла: СТУДЕНТЫ и ГРУППЫ. Первый файл должен содержать поля ФИО_СТУД, НОМЕР_СТУД_БИЛЕТА, СТИП, НОМЕР_ГРУППЫ, а второй — НОМЕР_ГРУППЫ, КУРАТОР, СРЕД_СТИП, СТУД_КОЛ. Большинство неудобств, связанных с обработкой данных, будут преодолены. Каждый из файлов будет содержать только недублируемую информацию, необходимости в динамических вычислениях суммарной информации не возникнет. Однако даже при таком переходе информационная система должна обладать некоторыми новыми особенностями, сближающими ее с СУБД.

Прежде всего, в системе должно быть указано, что она работает с двумя информационно связанными файлами, должна быть описана структура и смысл каждого поля (например, что НОМЕР_ГРУППЫ в файле Студенты и НОМЕР_ГРУППЫ в файле Группы означают одно и то же). Кроме того, необходимо понимать, что в ряде случаев изменение информации в одном файле должно автоматически вызывать модификацию во втором файле, чтобы содержимое этих файлов было согласованным. Например, если студен переводится из другого ВУЗа, то необходимо добавить запись в файл Студенты, а также соответствующим образом изменить поля СРЕД_СТИП и СТУД_КОЛ в записи файла Группы, описывающей группу, в которую переведен студент. ***Понятие согласованности данных является ключевым понятием баз данных.*** Фактически, если информационная система поддерживает согласованное хранение информации в нескольких файлах, можно говорить о том, что она поддерживает базу данных. Если же некоторая вспомогательная система управления данными позволяет работать с несколькими файлами, обеспечивая их согласованность, можно назвать ее системой управления базами данных. Это необходимый, но не достаточный признак системы управления базами данных. Уже только требование поддержания согласованности данных в нескольких файлах не позволяет обойтись библиотекой функций: такая система должна иметь некоторые собственные данные (метаданные) и даже знания, определяющие целостность данных.

Однако даже в нашем примере средствами прикладной программы неудобно реализовывать такие запросы как «выдать количество студентов в группе, в которой обучается Раевский Александр Ивано-

вич». Было бы гораздо проще, если бы СУБД позволяла сформулировать этот запрос на близком пользователям языке. Такие языки называются языками запросов к базам данных. Например, на языке SQL запрос можно было бы выразить в форме:

```
SELECT СТУД_КОЛ
FROM Студенты, Группы
WHERE ФИО_СТУД = "Раевский Александр Иванович"
AND Студенты.НОМЕР_ГРУППЫ = Группы.НОМЕР_ГРУППЫ
```

При наличии возможности обработки таких запросов СУБД позволит не задумываться о том, как будет выполняться этот запрос, и система сама выполнит просмотр файлов Студенты и Группы и выдаст соответствующий результат.

Прикладная система не обязана заботиться о корректности состояния базы данных во время сбоев в момент вставки или удалении данных, эти функции также входят в круг обязанностей СУБД.

Также, если опираться только на использование файлов, то для обеспечения корректности на все время модификации любого из двух файлов доступ других пользователей к этому файлу необходимо блокировать. Таким образом, занесение в файл сведений о новом студенте Раевском Александре Ивановиче существенно замедлит процесс получения информации о студенте Карасеве Алексее Александровиче, даже если они будут числиться в разных группах. При этом СУБД способна обеспечить гораздо более тонкую синхронизацию параллельного доступа к данным.

Кроме того, существует возможность значительно упростить структуру файла Группы, для этого необходимо удалить поля СРЕД_СТИП и СТУД_КОЛ из этого файла. При этом СУБД необходимо дополнить возможностью обработки простых статистических функций, таких как SUM – вычисление суммы и AVG – вычисление среднего значения из набора предложенных.

Таким образом, СУБД решают множество проблем, которые затруднительно или вообще невозможно решить при использовании программных файловых систем. При этом существуют следующие виды приложений: приложения, для которых вполне достаточно файлов; приложения, для которых необходимо решать, какой уровень работы с данными во внешней памяти для них требуется; приложения, для которых необходимо использовать системы управления базами данных.

Подводя итог, можно сформулировать основные принципы (положения), определяющие концепцию баз данных:

1) автономное безизбыточное хранение данных сложной структуры и значительного объема;

2) комплексное использование хранимой информации;

3) независимость программ обработки от физической структуры исходных данных.

Дополнительные положения концепции баз данных:

– БД есть отображение информационной модели предметной области;

– однократный ввод первичной информации;

– защита данных (авторизованный доступ — защита от катастрофического разрушения, криптография, ограничение целостности);

– реорганизация (развитие) БД по мере необходимости с минимальным влиянием на действующие программы [2].

Эти положения легли в основу большинства существующих определений баз данных и СУБД. Дж. Мартин в 1980 году сформулировал ***понятие БД как совокупности взаимосвязанных, хранящихся вместе данных*** при наличии такой организации и минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений; данные запоминаются и используются так, чтобы они были независимы от программ, использующих эти данные, а программы были независимы от способа и структуры хранения данных; для добавления новых или модификации существующих данных, а также для поиска данных в БД применяется общий управляющий способ [3].

СУБД — это специальная программа, предназначенная для обеспечения эффективного доступа к базе данных, используемая для предоставления только необходимой информации, обеспечения независимости от возможных изменений в структуре той части базы данных, которую не обрабатывает программа [3].

Основная особенность СУБД — это наличие процедур для ввода и хранения не только самих данных, но и описаний их структуры. Файлы, снабженные описанием хранимых в них данных и находящиеся под управлением СУБД, стали называть банки данных, а затем «Базы данных» (БД).

Приведенное выше определение отражает естественное развитие подхода к обработке структурированных данных и определяется в основном требованием повышения эффективности функционирования программного обеспечения. Принцип обеспечения независимости программ от физической организации данных был в то время определяющим, а простота доступа понималась как возможность простого обращения к БД из программы, написанной на стандартном языке программирования.

Вторым по важности был неявно обозначенный выше принцип информационного моделирования некоторой предметной области в виде БД.

На более поздних этапах развития процессов обработки данных на ЭВМ в связи с применением персональных компьютеров конечными пользователями все возрастающее значение приобретает именно эта способность отображения в базах данных информационной модели предметной области и обеспечения непосредственного доступа к базам данных без предварительного программирования. Поэтому на первый план выступают принципы автономного хранения данных сложной структуры и простого авторизованного доступа (комплексного использования), причем под «простым» понимается доступ к БД без предварительного программирования, т.е. доступ конечных пользователей.

Таким образом, необходимость применения концепции баз данных обусловлена следующими причинами [2]:

1) развитием подхода к обработке данных — от вычислительных задач к информационным; объединением последних в комплексы (подсистемы) с постоянным их развитием, включая расширение состава задач и ориентирование на широкий круг конечных пользователей;

2) противоречием между позадачным подходом в использовании исходных данных и требованием их эффективной актуализации;

3) стремлением отобразить в системе хранимых данных информационную модель определенной предметной области.

Контрольные вопросы

1. Дайте определение файла с точки зрения прикладной программы.
2. Опишите недостатки позадачного подхода в использовании исходной информации.
3. Назовите причины вызвавшие появление баз данных.
4. Приведите определения БД и СУБД.
5. Назовите основные положения концепции БД.

2. ФУНКЦИИ СУБД. АРХИТЕКТУРА ПРЕДСТАВЛЕНИЯ ИНФОРМАЦИИ В КОНЦЕПЦИИ БД

2.1. Функции СУБД

За время эволюции СУБД был определен не только круг решаемых ими задач, но и перечень основных функций, поддержка которых в системе необходима для решения этих задач. В каждой конкретной СУБД имеется свой определенный набор функций, который варьируется в зависимости от сложности системы. Однако для большинства современных СУБД, относящихся к классу коммерческих систем, очевидна поддержка нескольких базовых функций, к числу которых можно отнести следующие [1]:

1) управление данными во внешней и оперативной памяти;
2) управление транзакциями;
3) журнализация изменений БД;
4) поддержка языков доступа к данным;
5) обеспечение безопасности базы данных.

Управление данными во внешней и оперативной памяти (на жестком диске) включает не только обеспечение необходимых структур внешней памяти для хранения данных, непосредственно входящих в БД, и для служебных целей, но и возможность системы хранить изменяемую часть БД во время работы с ней в оперативной памяти. Самым простым и эффективным способом при оптимизации доступа к данным на физическом уровне является индексирование данных.

В некоторых типах СУБД активно используются возможности существующих файловых систем, в других — работа производится вплоть до уровня устройств внешней памяти [1]. Большинство современных СУБД поддерживает собственную систему именования объектов БД.

СУБД при работе с БД значительного размера должна обеспечивать обмен данными между БД и пользователем посредством использования буферов оперативной памяти, что позволяет увеличить скорость обработки данных. Некоторые современные СУБД позволяют при работе с базой данных полностью загружать ее в оперативную память компьютера.

Управление транзакциям

Транзакция — это последовательность операций над БД, рассматриваемых СУБД как единое целое. Понятие транзакции необходимо для поддержания логической целостности БД. Если вспомнить наш

пример информационной системы с файлами СОТРУДНИКИ и ОТДЕЛЫ (см. подраздел 1.3), то единственным способом не нарушить целостность БД при выполнении операции приема на работу нового сотрудника является объединение элементарных операций над файлами СОТРУДНИКИ и ОТДЕЛЫ в одну транзакцию. Таким образом, поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД. Но понятие транзакции гораздо более важно в многопользовательских СУБД [1].

В области обработки транзакций существует следующая классификация [4]:

Первое поколение. Единые монолитные системы, основанные на примитивных моделях терминалов для взаимодействия с пользователем.

Второе поколение. Поддержка продуктов многих поставщиков, интеллектуальные клиентские системы, поддержка множества систем баз данных, как правило при помощи протоколов двухфазной фиксации.

Третье поколение. Поколение новых систем, более тесно связанное с потребностями моделирования бизнес-процессов.

Любая транзакция основана на некотором наборе принципов, называемых ACID (Atomicity, Consistency, Isolation, Durability). Смысл этих принципов заключается в следующем [5]:

Атомарность (Atomicity). Как было указано выше транзакция представляет собой некоторый набор действий, при этом система обеспечивает их выполнение по принципу «все или ничего» – либо выполняются все действия, и транзакция фиксируется, т.е. СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо не выполняется ни одного и транзакция завершается аварийно, т.е. ни одно из этих изменений никак не отражается на состоянии БД.

Целостность (Consistency). Каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, что делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. Понятие транзакции позволяет разработчику декларировать точки целостности, а системе производить их верификацию с помощью предоставляемым приложением проверок.

Изолированность (Isolation). Поскольку транзакция обновляет совместно используемые данные, то для них могут временно нарушаться условия целостности. Данные, для которых такие нарушения возникли, не должны быть видимы другим транзакциям, пока произ-

водимые обновления не будут зафиксированы. Система должна обеспечить для каждой транзакции иллюзию того, что она выполняется изолировано, как будто другие транзакции либо уже завершились до ее начала, либо начнут выполняться после ее фиксации.

Здесь следует отметить, что при соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД. С управлением транзакциями в многопользовательской СУБД связаны важные понятия сериализации транзакций и сериального плана выполнения смеси транзакций. ***Под сериализаций параллельно выполняющихся транзакций,*** согласно [1], понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. ***Сериальный план выполнения смеси транзакций —*** это такой план, который приводит к сериализации транзакций. Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Согласно [1], в современных СУБД наиболее распространены алгоритмы сериализации транзакций, основанные на синхронизационных захватах объектов БД. При обеспечении принципа сериализации транзакций в СУБД возможны ситуации конфликтов между несколькими транзакциями во время доступа к объектам БД. В этом случае для поддержания целостности данных и сериализации необходимо выполнить откат транзакции (ROLLBACK).

Долговременное хранение (Durability). Если транзакция зафиксирована, то ее результаты должны быть долговечными. Новые состояния всех объектов должны сохраняются в специальной области БД.

Существуют многочисленные модели транзакций, поддерживающие эти принципы, – от простейших, например плоских транзакций, до более сложных, таких как вложенные и многозвенные транзакции. Более подробную информацию о моделях транзакций и системах управления транзакциями можно найти в [6. глава 20].

Журнализация изменений БД

Главным критерием при выборе СУБД является ***надежность хранения информации —*** возможность СУБД восстановить последнее согласованное состояние БД после какого-либо сбоя.

Различают два вида сбоев: программный и аппаратный. Аппаратные сбои в свою очередь можно разделить на мягкие сбои, возникаю-

щие, например, при аварийном выключении питания, и жесткие сбои, при которых возможна частичная или полная потеря информации. Под программным сбоем можно подразумевать аварийное завершение работы СУБД.

В результате программных и аппаратных сбоев во время работы пользователя с БД некоторые транзакции могут остаться незавершенными. Для восстановления БД необходимо хранить информацию об изменениях, производимых в БД, т.е. журнал изменений БД.

Журнал — это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД [1].

Идеология формирования журнала основывается на необходимости соблюдения принципа упреждающей записи об изменениях БД в журнал (Write Ahead Log – WAL), т.е. информация об изменениях данных в БД в журнале должна появиться до того, как произойдут эти изменения. Если в СУБД корректно ведется журнал БД, то с его помощью можно восстановить базу после любого сбоя (естественно, если в результате сбоя не утерян сам журнал).

Основным способом восстановления БД является индивидуальный откат транзакций.

При мягком сбое в БД могут находиться данные, модифицированные транзакциями, не закончившимися к моменту сбоя, и могут отсутствовать данные, модифицированные транзакциями, которые к моменту сбоя успешно завершились. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций [1]. Процесс восстановления производится путем отката незавершенных транзакций, после чего повторно выполняются операции завершенных транзакций, результаты которых не отражены в БД.

При жестком сбое для восстановления БД необходимо использовать кроме журнала и архивную копию БД, находящейся в согласованном состоянии. Принцип восстановления БД состоит в том, что по архивной копии и, следуя журналу, должны быть, отработаны все завершенные транзакции.

Поддержка языков БД

Основной обязанностью любой СУБД, помимо хранения данных, является возможность оперировать этими данными. Для работы с БД используются синтаксические конструкции, в целом называемые язы-

ками баз данных. В большинстве ранних СУБД изначально поддерживалось несколько типов языков. Можно выделить два языка — язык манипулирования данными DML (Data Manipulation Language) и язык определения схемы БД SDL (Schema Definition Language). С помощью SDL определялась логическая структура БД, DML содержал набор операторов для манипулирования данными. В более поздних СУБД был образован единый интегрированный язык, с помощью которого можно полноценно управлять БД, начиная от ее создания и заканчивая выводом структурированной информации в ответ на запрос, написанный на этом языке. Одним из таких языков, используемый в большинстве реляционных СУБД является язык SQL (Structured Query Language).

Язык SQL позволяет выполнять функции языков SDL и DML, с его помощью можно сгенерировать схему БД и манипулировать данными. С помощью языка SQL можно создавать сложные конструкции, определяя, в том числе и ограничения целостности БД.

Более подробно стандарты и функции языка SQL, а также других языков манипулирования данными будут рассмотрены в разделе 7 данного пособия.

Обеспечение безопасности базы данных

Защита информации в информационных системах является серьезной и кропотливой задачей. На самом элементарном уровне ее решение сводится к обеспечению выполнения двух фундаментальных принципов: проверки полномочий пользователя – *санкционирование доступа* и проверку подлинности – *аунтификацию* [6].

Проверка полномочий пользователя основывается на определении и последующей проверке для каждого пользователя набора санкционированных действий, которые он может выполнять по отношению к определенным объектам базы данных.

В большинстве СУБД задание и проверка полномочий определяется внутренними средствами системы, одним из способов задания полномочий является использование операторов Grant и Revoke языка SQL.

Определяются следующие уровней доступа пользователей к объектам БД:

- создание объекта БД;
- изменение;
- чтение;
- удаление;
- администрирование (определяет полный доступ к объекту).

Сведения о полномочиях пользователя находятся защищенной области базы данных.

Проверка подлинности заключается в достоверном подтверждении того, что пользователь, выполняющий санкционированные действия, действительно является тем, за кого себя выдает [6].

В любой СУБД должна поддерживаться модель проверки подлинности, которая может обеспечить надежную верификацию идентификаторов, предъявляемых пользователями. Обычно контроль подлинности пользователя, работающего с базой данных, заключается в сопоставлении вводимых пользователем имени и пароля при входе в систему с эталонными идентификаторами пользователя, сохраненными в БД.

Одним из способов безопасного хранения данных является использование *модели многоуровневой безопасности данных* [6], такая система безопасности означает, что в системе хранится информация, относящаяся к разным классам безопасности, при этом пользователь может иметь доступ только к уровню определенном конкретно для него и нижним уровням.

Многоуровневая безопасность в отношении БД может строится на основе *модели Белла – ЛаПадула* [6], при этом объекты базы данных подвергаются классификации (например, особо секретно, секретно, конфиденциально, для общего пользования), а каждый пользователь причисляется к одному из уровней допуска к классам объектов.

Механизмы обеспечения безопасности данных постоянно совершенствуются разработчиками СУБД и являются на сегодняшний день одним из перспективных направлений в развитии информационных технологий. Организацию политики безопасности в СУБД MS Access мы рассмотрим в главе 10 настоящего пособия.

Рассмотренные пять основных функций СУБД являются базовыми, при этом в каждой конкретной системе их реализация обладает определенной спецификой и зачастую является закрытой и тщательно скрываемой от конкурентов сложной информационной технологией.

2.2. Архитектура представления информации в концепции БД

В основе теории БД лежит технология создания различных уровней моделей предметной области, базирующихся на понятии представления данных. Выделяют три уровня представления информации (рис. 2.1):

физическое представление **—** размещение физической структуры и значений хранимых данных в памяти компьютера (во внешней и в оперативной);

концептуальное представление **—** логическая структура БД, формальное описание предметной области в терминах БД, представляющее описание объектов, с указанием взаимосвязей между ними, без определения методов и способов их физического хранения;

внешнее представление **—** часть структуры БД, используемая пользователем для выдачи информации в конкретном приложении.

С помощью СУБД можно наглядно реализовать все три уровня представлений и тем самым обеспечить соблюдение основных принципов концепции БД. Хранение описания физической структуры БД позволяет СУБД обеспечить работу с конкретными данными при передаче ей имен данных, что обеспечивает независимость программ, с помощью которых были организованы запросы, от способа размещения данных в памяти.

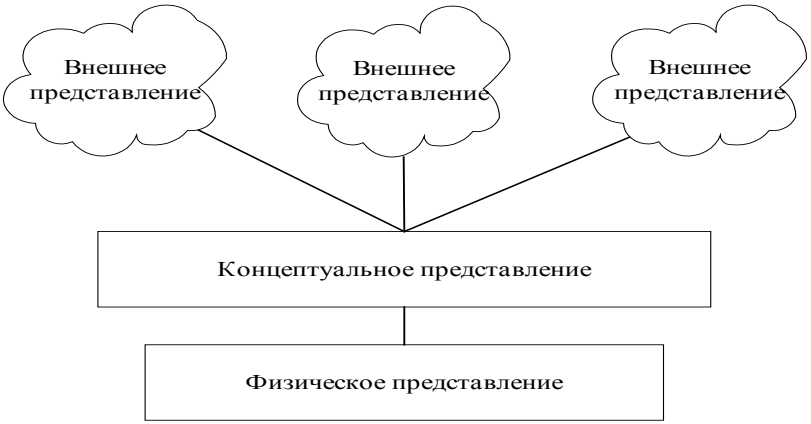

Рис. 2.1. Уровни представлений БД

С другой стороны, соотнесение и отображение с помощью СУБД любого внешнего представления с общей концептуальной моделью как раз и является основой обеспечения комплексного использования хранимых данных [2]. Возможность разделения представлений и принцип автономного хранения и ведения данных является основой централизованного хранения информации БД.

Наглядно пояснить понятия концептуального, физического и внешних представлений можно на следующих примерах.

Пример 2.1

Концептуальное представление данных можно отобразить в виде упрощенной модели предметной области «Успеваемость студентов ВУЗа», для чего достаточно иметь сведения о студентах и оценке, по-

лученной студентом по конкретной дисциплине в конкретном семестре. Обычно для иллюстрации процесса проектирования концептуальной модели используют графическое представление информационных элементов: объектов и взаимосвязей. В нашем примере стрелкой определяется связь между двумя элементами модели, указывая, что каждой записи (строке) в таблице СТУДЕНТЫ, будет соответствовать несколько записей в таблице УСПЕВАЕМОСТЬ (рис. 2.2).

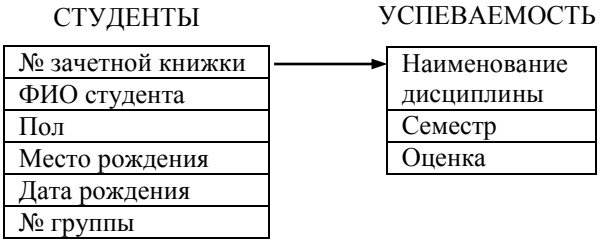

Рис. 2.2. Концептуальное представление информации об успеваемости студентов ВУЗа

Спроектированная концептуальная модель выбранной предметной области служит основой для создания физической модели базы данных в идеологии конкретной СУБД.

Пример 2.2

При формировании физического представления, т.е. при создании реальных таблиц БД (с определением типов данных, полей связи и другой сопроводительной информации) в наших таблицах будут созданы поля ID_Stud, по которым будут связаны эти таблицы (рис. 2.3).

Таблица 1

ID_Stud	№ зачетной книжки	ФИО студента	Пол	Место рождения	Дата рождения	№ группы

Таблица 2

ID_Stud	Наименование дисциплины	Семестр	Оценка

Рис. 2.3. Физическое представление информации о студентах ВУЗа

Следует отметить, что физическое представление в идеологии одной СУБД может отличаться от аналогичного представления в другой СУБД, что вызвано возможными различиями в определении типов данных конкретных полей, так, в ранних СУБД отсутствуют типы

данных, новейших систем управления базами данных, такие как гиперссылки, OLE-объекты и др. Кроме того, существует ряд СУБД, не поддерживающих символы кириллицы в именах полей и наименованиях таблиц. По этой причине следует с особым вниманием подходить к проектированию БД, учитывая ограничения, накладываемые разработчиками СУБД.

Пример 2.3

Внешнее представление. Создавая приложение, разработчик определяет перечень таблиц и полей этих таблиц, необходимых для ввода или вывода данных. На рис. 2.4 представлен внешний вид отчета и внешнее представление БД, необходимое для его создания.

Информация об успеваемости студентов за 3-й семестр

ФИО студента	Дисциплина	Оценка

ФИО студента	Дисциплина	Оценка	Семестр

Рис. 2.4. Внешний вид отчета об успеваемости студентов ВУЗа и соответствующее внешнее представление

Описание концептуального и соответствующего ему физического представления (описание структуры БД) хранится автономно, называется ***схемой БД*** и создается разработчиком до того, как начнет наполняться БД.

Описание подмножества концептуального представления, которое соответствует внешнему представлению для некоторого приложения (описание части структуры БД, доступной программе обработки), называется ***подсхемой*** [2]. С помощью схемы и подсхемы был достигнут уровень развития СУБД, при котором обеспечивается принцип независимости данных от прикладных программ, а также неизменность внешних представлений при изменении структуры данных, которые не нарушают функциональных возможностей соответствующих прикладных программ.

Разработчику, создавая пользовательское приложение, оперирующее данными, находящимися в спроектированной БД, для получения требуемого результата, достаточно определить требуемое внешнее представление как подмножество концептуального. Более подробно

основы применения представлений и соответствующих им моделей данных подробно описаны в [7]. В современных СУБД существуют средства графической реализации подсхем, являющихся основой внешних представлений проектируемого разработчиком пользовательского приложения.

Таким образом, одна из главных задач разработчика (администратора) базы данных состоит в создании концептуальной модели предметной области, обеспечивающей концептуальное представление данных и выборе СУБД для ее практической реализации.

Контрольные вопросы

1. Перечислите и кратко охарактеризуйте основные функции СУБД.
2. Дайте определения представлений данных.
3. Дайте определение понятий схемы и подсхемы данных.

3. МОДЕЛИ ДАННЫХ

3.1. Дореляционные модели данных

Какие бы СУБД мы не рассматривали, в основе каждой лежит использование определенной модели (или моделей) данных, отражающих связи между объектами БД.

За время эволюции БД сформировались понятия о следующих видах моделей данных:

- линейной;
- иерархической;
- сетевой;
- реляционной;
- объектно-ориентированной.

Линейная, иерархическая и сетевая модель относятся к классу так называемых дореляционных моделей данных, явившихся базисом для создания дореляционных СУБД (подробно свойства этих СУБД рассмотрим в разделе 10).

Можно сформулировать некоторые основные характеристики дореляционных СУБД, описанные в [1]:

- эти системы активно использовались в течение многих лет, дольше, чем используется какая-либо из реляционных СУБД;
- все ранние системы не основывались на каких-либо абстрактных моделях. Абстрактные представления ранних систем появились позже на основе анализа и выявления общих признаков у различных систем;
- в ранних системах доступ к БД производился на уровне записей. Пользователи этих систем осуществляли явную навигацию в БД, используя языки программирования, расширенные функциями СУБД. Интерактивный доступ к БД поддерживался только путем создания соответствующих прикладных программ с собственным интерфейсом;
- навигационная природа ранних систем и доступ к данным на уровне записей заставляли пользователя самого производить всю оптимизацию доступа к БД, без какой-либо поддержки системы;
- после появления реляционных систем большинство ранних систем было оснащено «реляционными» интерфейсами. Однако в большинстве случаев это не сделало их по-настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

Появление технология организации БД послужило толчком для дальнейшего развития и совершенствования автоматизированной ком-

плексной обработки структурированной информации, были сформулированы особенности концепции БД:

1) ***информационно описывается совокупность объектов некоторой предметной области*** [2]. Это могут быть объекты разнородных типов (документы, товары, сотрудники, организации и др.). Идея такого подхода заключается в том, что объекты выбранной предметной области должны обладать определенными свойствами, (значениями, параметрами, характеристиками и т.п.), с помощью которых пользователь мог бы получить полной представление о выбранном объекте. Отметим, что для разных объектов значения разных параметров могут относиться к одному типу данных (символьному, логическому и т.д.) и выбраны эти значения могут быть из одного множества (определены на одном домене);

2) ***при автоматизированном моделировании предметной области должны выполняться следующие требования:***

– каждому параметру объекта предметной области соответствует данное, значению параметра у конкретного объекта — значение данного в записи, соответствующей этому объекту. Данное-идентификатор объекта (может быть составное данное), — параметр, по которому однозначно можно определить объект, называется ключевым данным записи;

– перечень однотипных объектов с указанием их характеристик соответствует таблице, в которой одному объекту соответствует одна либо несколько строк, соответственно параметру объекта соответствует столбец таблицы;

Необходимость хранения и эффективного использования информационной модели предметной области явилась одной из основных причин возникновения концепции БД и использования СУБД [2].

Начало формирования понятия структур данных приходится на период, предшествующий применению СУБД, в то время широко использовались простые алгоритмические языки программирования, с помощью которых пользователь мог формировать простые файлы, содержащие так называемые ***линейные структуры*** данных, т.е. простые последовательные записи. Связи между такими файлами устанавливались средствами языков программирования, что было чрезвычайно неэффективно.

Под структурой данных будем понимать совокупность информационных элементов и связей между ними. Под моделью данных будем понимать соответствующих тип структуры данных и типовые операции по управлению данными в этих структурах [2].

Следует также заметить, что когда говорят о структуре данных как о модели данных, то имеют в виду логическую структуру, под которой понимают представление информационных элементов и связей между ними вне зависимости от способа их размещения в памяти компьютера [2]. Структуру же, в которой определены поля связи, типы данных, технология размещения данных, принято называть физической структурой.

Перейдем к описанию типовых моделей данных.

3.2. Линейная модель данных

Линейная структура данных, как отмечалось в подразделе 3.1, представляет собой плоскую таблицу. Для линейной структуры характерны следующие свойства [2]:

- элементами линейной структуры являются простые данные, разделение которых на составляющие не имеет смысла;
- каждое данное (поле) имеет имя (идентификатор) и множество возможных значений, задаваемое словарем, диапазоном или правилом формирования;
- множество данных, составляющих линейную структуру, описывает множество однотипных объектов;
- все экземпляры линейной структуры (записи) однородны, при этом порядок следования данных во всех экземплярах структуры один и тот же, а максимальный размер и тип данного одного имени во всех экземплярах структуры одинаковы. Естественно, что разные данные могут иметь различные размеры и типы.

Для линейной структуры характерно наличие ключевых данных, представляющих собой одно либо несколько полей данных, значения которых однозначно определяют каждый экземпляр структуры. Вообще говоря, весь диапазон данных линейной структуры уже однозначно определяет объект, однако в данном случае имеется в виду оптимальный минимум данных. Набор таких данных называется ***первичным ключом.*** Если в состав первичного ключа входит более одного поля данных, такой ключ называется ***составным первичным ключом.***

На рис. 3.1 представлена простая линейная структура, содержащая однородные данные, первичным ключом в которой является ***ID_Stud,*** однако поле ***№ зачетной книжки*** тоже однозначно определяет студента, такое поле называется ***альтернативным первичным ключом***. Заголовок линейной структуры называется схемой.

ID_Stud	№ зачетной книжки	ФИО студента	Пол	Место рождения	Дата рождения	№ гр.
1	1992412-11	Карасев А.А.	М	г. Чита	27.08.75	412-1
2	1992432-11	Данилов О. В.	М	г. Алма-Ата	27.08.75	432-1
3	1992432-12	Раевский А. И.	М	г. Бишкек	20.05.75	432-1

Рис. 3.1. Простая линейная структура данных СТУДЕНТЫ

Над линейными структурами возможно применение основных операций по управлению данными:

- ***вставки*** — добавления новых записей в структуру;
- ***удаления***— удаления записей из структуры;
- ***замены*** — изменения значений данных в указанных записях;
- ***выборки*** — отбора конкретных экземпляров структуры, необходимых для дальнейшей обработки.

Для создания более сложных моделей данных (за исключением объектно-ориентированной) в качестве простой составляющей используют линейные структуры данных, при этом необходимо учитывать типы взаимосвязей, возникающие между разными простыми структурами. Принято различать взаимосвязи «один-к-одному» – (1:1), «один-ко-многим» (1:М) и «многие-ко-многим» (М:М). Связи между структурами данных являются неотъемлемой частью концептуальной модели разрабатываемой БД.

3.3. Иерархическая модель данных

Иерархическая структура имеет место при описании нескольких разнотипных, взаимосвязанных объектов, находящихся в строгой иерархии. Такая структура состоит из узлов (элементов, сегментов) — совокупности атрибутов данных, описывающих объект, и ветвей — связей между типами объектов.

На рис. 3.2 представлена иерархическая структура данных, стрелка указывает путь в иерархии от старшего к подчиненному. Каждому экземпляру структуры СТУДЕНТЫ соответствует несколько экземпляров структуры ПЛАТА ЗА ОБУЧЕНИЕ и УСПЕВАЕМОСТЬ, кото-

рой в свою очередь соответствует несколько экземпляров структуры КОНТРОЛЬНЫЕ ТОЧКИ.

Такую иерархическую структуру принято также называть древовидной. Наивысший узел (в нашем примере СТУДЕНТЫ) называется корнем. Зависимые узлы располагаются на более низких уровнях дерева. Узел, каждому экземпляру которого можно поставить в соответствие несколько экземпляров другого узла, называется старшим элементом или родителем. Следующий за ним элемент в структуре называется подчиненным элементом или потомком, который в свою очередь может иметь несколько типов подчиненных элементов и быть старшим по отношению к ним. Тип связи в иерархической древовидной структуре определяется как 1:M — один-ко-многим.

Рис. 3.2. Иерархическая древовидная структура данных

Если на структуру наложено ограничение, согласно которому каждому экземпляру подчиненного узла обязательно должен соответствовать экземпляр родительского узла, то такую иерархию называют жесткой. Если же в структуре допускается отсутствие экземпляров роди-

тельского узла для существующих подчиненных, то такую иерархию называют формальной.

Элементы, не имеющие подчиненных, называют ***концевыми*** элементами или листьями. Путь от концевого элемента до корневого называют ***ветвью***, а максимальное количество элементов в самой «длинной» ветви — ***рангом*** структуры. Корневой элемент определяет первый уровень структуры; элементы, непосредственно связанные с корневым, — второй уровень и т.д. [2].

Формально дерево можно определить как иерархию элементов с попарными связями, для которой выполняются следующие правила:

1) самый верхний уровень имеет только один узел или корневой сегмент;

2) каждый узел состоит из одного либо нескольких элементов данных, описывающих объект в узле;

3) каждый низший уровень может содержать зависимые узлы, и тогда узел, находящийся на предыдущем уровне, называется исходным (родителем). Зависимые узлы могут добавляться как вертикально, так и горизонтально;

4) каждый узел, находящийся на втором уровне, соединен с одним и только одним узлом на уровне первом и т.д.;

5) исходный уровень может иметь в качестве зависимых любое количество порожденных узлов;

6) доступ к каждому узлу за исключением корневого происходит через исходный узел.

Операции по управлению данными в иерархической модели данных:

включение — добавление экземпляров в элемент структуры, аналогична операции вставки в линейных структурах, с ограничением на вставку экземпляров при отсутствии старшего (для жесткой иерархии);

замена (обновление) — операция, аналогичная операции замены в линейных структурах, однако для родительских элементов древовидной структуры для обеспечения принципа жесткой иерархии необходимо автоматическое обновление полей связей экземпляров подчиненных элементов. В некоторых иерархических СУБД допускается формальное отсутствие ключевых полей в подчиненных элементах: ключи старших «мигрируют» в подчиненные элементы;

удаление —каскадное удаление всех экземпляров подчиненных элементов при удалении соответствующих им старших при жесткой иерархии;

выборка — возможность выборки *подобного экземпляра* в отличие от операции выборки в линейных структурах, т.е. операция позволяет произвести чтение следующего экземпляра того же элемента

структуры. Обеспечивается также выборка подобного экземпляра в пределах исходного. Такая выборка может быть осуществлена только после выборки экземпляра старшего элемента, а читаются последовательно однотипные подчиненные, но только связанные с выбранным «старшим» [2]. Возможна также выборка следующего экземпляра в иерархической последовательности, т.е. выборка данных по правилу перехода по дереву — «сверху вниз – слева направо».

Основными достоинством иерархической модели данных является простота понимания и использования, кроме того достигается определенный уровень в обеспечении независимости данных. К недостаткам модели можно отнести сложность механизма обеспечения связей типа «многие-ко-многим» и, как следствие, невозможность легко представить в виде древовидной структуры большинство предметных областей, а также отсутствие гибкости при выполнении операций по манипулированию данными.

3.4. Сетевая модель данных

В сетевой модели объекты предметной области объединяются в сеть. Элементами такой сетевой структуры являются линейные структуры данных. Иерархическая структура является частным случаем сетевой. Связи в сетевой структуре определяются так же, как и в иерархической. Заметим, что при определении связей в сетевой структуре допустимы следующие положения:

– в сетевой структуре может быть несколько главных элементов (корней) или главный элемент вообще может отсутствовать;

– допускается наличие более одной связи между двумя элементами структуры;

– подчиненный элемент может иметь более одного старшего;

– допускаются циклические связи;

– возможно наличие связей между экземплярами одного и того же элемента структуры.

Поясним различие между понятиями элемент структуры (типом записи) и экземпляр элемента структуры (экземпляром записи). Студенты – является типом структуры, а «1112 Иванов И.И. 1987… » является экземпляром элемента структуры. Таким образом в БД может иметься один или несколько экземпляров элемента структуры или, что тоже верно, в БД может существовать любое множество экземпляров записи некоторого типа.

На рис. 3.3. приведен пример простой сетевой структуры.

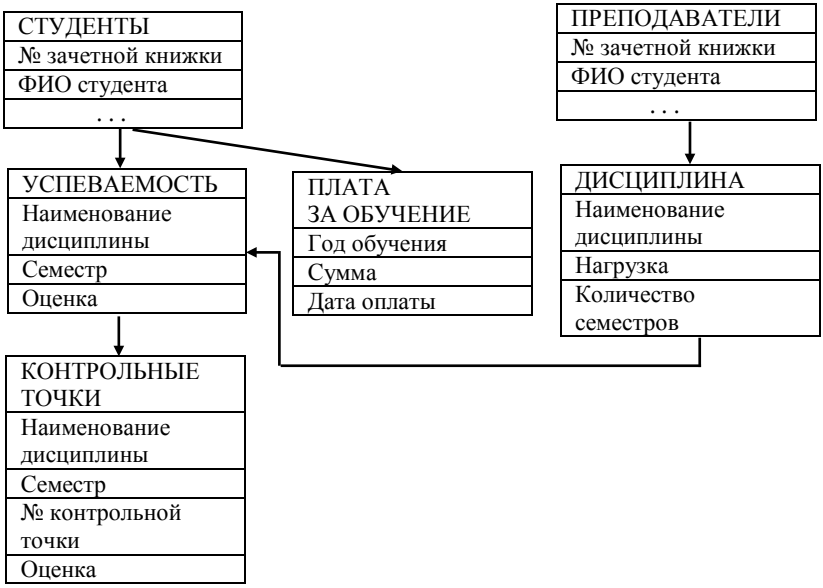

Рис. 3.3 Сетевая структура данных

В этом примере мы расширили иерархическую структуру, предложенную в предыдущем подразделе, добавив два элемента: ПРЕПОДАВАТЕЛИ и ДИСЦИПЛИНА, предположив, что один преподаватель может читать лекции по нескольким дисциплинам (связь 1:М). Элемент ДИСЦИПЛИНА также связан с элементом УСПЕВАЕМОСТЬ (1:М). Таким образом, в элементах УСПЕВАЕМОСТЬ и КОНТРОЛЬНЫЕ ТОЧКИ можно опустить поля НАИМЕНОВАНИЕ ДИСЦИПЛИНЫ, поскольку при установке соответствующих связей, автоматически добавятся ключевые поля старших структур в дочерние.

Операции по обработке данных в сетевой модели аналогичны соответствующим операциям для иерархической модели.

Основным достоинством сетевой модели является ее универсальность и возможность реализации связей многие-ко-многим. Главным

недостатком является ее сложность, разработчик должен детально знать структуру всей БД, даже той части, которую не затрагивает его программа, поскольку необходимо осуществлять навигацию среди различных элементов структуры.

Следующим шагом в развитии моделей данных использование реляционной модели данных при проектировании БД, которая из-за своей простоты и универсальности заняла прочное место в современных системах управления базами данных.

Контрольные вопросы

1. Перечислите дореляционные модели данных.
2. Дайте определения физической и логической структуры данных.
3. Назовите правила построения иерархических структур.
4. Назовите и кратко охарактеризуйте типовые операции по управлению данными в дореляционных структурах.

4. РЕЛЯЦИОННАЯ МОДЕЛЬ

4.1. Основные понятия реляционной модели

4.1.1. Общие сведения

Использование реляционной модели данных было предложено доктором Э.Ф. Коддом в 1970 г. Реляционная модель основана на теории отношений [7], при проектировании БД применяются строгие методы, построенные на нормализации отношений (глава 6, настоящего пособия). Доктор Кодд, в своей статье [8] отмечает, что реляционная модель данных обеспечивает ряд возможностей, которые делают управление БД и их использование относительно легким, устойчивым по отношению к ошибкам и предсказуемым. Наиболее важные характеристики реляционной модели заключаются в следующем [9]:

- реляционная модель описывает данные с их естественной структурой, не добавляя каких-либо дополнительных структур, необходимых для машинного представления или для целей реализации;
- она обеспечивает математическую основу для интеграции выводимости, избыточности и непротиворечивости отношений;
- реляционная модель позволяет добиться реальной независимости данных от их физического представления, от связей между данными и от способов реализации, связанных с эффективностью и подобными заботами.

Благодаря доктору Э.Ф. Кодду, компания IBM в начале 70-х годов начала разработку нескольких коммерческих реляционных СУБД. В начале 80-х годов такие гиганты информационной индустрии, как Oracle Corporation, Ingres Corp., IBM и других более мелких организаций предложили ряд систем, наглядно демонстрирующих возможность применения реляционной модели для разработки баз данных, хранящих информацию о любой предметной области, а также возможность реализовывать на их основе гибкие пользовательские приложения.

Единственным ограничением для широкомасштабного внедрения реляционных СУБД являлась низкая производительность средств вычислительной техники. Появление и стремительное распространение персональных компьютеров, а также простота управления реляционной БД способствовали быстрому расширению рынка реляционных СУБД и признанию таких систем разработчиками и пользователями приложений, построенных с использованием этих БД.

4.1.2. Смысл понятий реляционной модели

Основными понятиями структурной части баз данных, строящихся на реляционных моделях, являются:

- отношение;
- тип данных;
- домен;
- атрибут;
- кортеж;
- первичный ключ;

Смысл этих понятий наглядно поясним на примере отношения СТУДЕНТЫ, представленного на рис. 4.1.

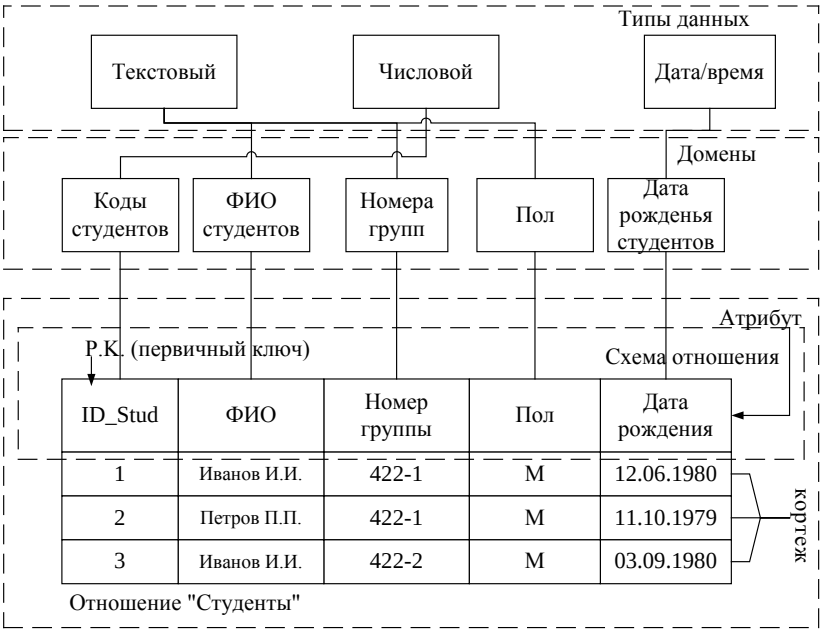

Рис. 4.1. Отношение СТУДЕНТЫ

В реляционной модели данные представляются в виде плоских таблиц, называемых отношениями. Столбец таблицы называется атрибутом или полем, строка — кортежем отношения. Рассмотрим более подробно основные понятия и определения, используемые в реляционной модели данных.

4.1.3. Отношение, схема отношения, кортеж

Схема отношения состоит из названий атрибутов и типов данных, на которых определены эти атрибуты. Можно сказать, что схема отношения есть конечное множество имен атрибутов, которым ставится в соответствие определенный тип данных (или домен, если СУБД поддерживает это понятие). Степень схемы отношения есть мощность этого множества. Степень или арность отношения СТУДЕНТЫ равна пяти, то есть это отношение является 5-арным. Таким образом, ***схема БД есть набор схем отношений.***

Отношение есть линейная структура данных, состоящая из множества кортежей, соответствующих одной схеме отношения [1]. Схему отношения называют заголовком, а совокупность кортежей отношения — телом отношения. Кортеж отношения (запись) описывает часть экземпляра объекта предметной области (ПрО) или, если объект ПрО характеризуется одним отношением, в одном кортеже отражается полная характеристика экземпляра объекта.

Таким образом, реляционная база данных состоит из набора взаимосвязанных отношений, имена которых совпадают с именами схем отношений в схеме БД [1]. При проектировании базы данных сначала определяют схемы отношений, после чего заносят данные. В некоторых СУБД после определения схемы отношения нельзя ни удалить и ни переименовать, ни один из его атрибутов. Однако можно удалять отношения, менять их названия, менять типы данных атрибутов. Структурное изменение схем отношений БД называют также эволюцией базы данных.

4.1.4. Тип данных

Типам данных в реляционной модели можно сопоставить типы данных, используемые в языках программирования. Все атрибуты в отношении должны быть определенного типа.

Данные, хранящиеся в реляционных БД, могут быть следующего типа:

символьные (текстовые);

числовые;

логические;

дата/время;

В некоторых СУБД введены дополнительные типы данных, например, в СУБД MS ACCESS используется тип данных объекта OLE — в полях такого типа можно хранить графические изображения, файлы документов и электронные таблицы, а также другие подобные объекты.

4.1.5. Домен

Домен — есть множество допустимых значений атрибута определенного типа. Это понятие характерно для баз данных, аналогично подтипам в языках программирования высокого уровня, домен может быть определен на основе конкретного типа данных.

Домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат «истина», то элемент данных является элементом домена [1].

Например, домен «Дата рождения» в нашем примере определен на базовом типе дата/время, но в число его значений могут входить только даты, которые могут отображать только дату без отображения времени. Атрибут ПОЛ текстового типа, определенный на одноименном домене, может принимать только два значения: М или Ж.

В СУБД, использующих понятие домена, атрибуты отношения считаются сравнимыми в том и только том случае, если эти атрибуты определены на одном домене. В примере (рис 3.4) значения доменов «Номера групп» и «Пол» относятся к текстовому типу, но не являются сравнимыми.

4.2. Свойства отношений

4.2.1. Уникальность кортежей отношения

Отношениям, как основной единице построения реляционных БД, присущи определенные свойствами и правила заполнения тела отношения сведениями об экземплярах объекта предметной области.

Главным отличием реляционного отношения от плоской таблицы заключается в том, что отношение (в классическом его понимании) не может иметь дубликатов кортежей. Поскольку отношение есть множество кортежей, а каждое множество не должно включать одинаковых элементов, то уже этого достаточно для обеспечения уникальности кортежей. Обеспечить это требование возможно с помощью первичного ключа — набора атрибутов, значения которых однозначно определяют кортеж отношения.

В современных СУБД для обеспечения свойства уникальности записей в каждой таблице БД дополнительно вводится идентификатор записи (инкремент, счетчик и т.д.) — атрибут, значение которого является уникальным для каждой записи отношения БД. Наличие такого

идентификатора записи необходимо в связи с возможным отсутствием первичного ключа в некоторых создаваемых разработчиками отношениях. Обычно же в состав первичного ключа включается минимальный набор атрибутов отношения, являющегося идентификатором объекта. Наличие первичного ключа необходимо для определения связей между отношениями и, как следствие, для обеспечения целостности данных.

4.2.2. Отсутствие упорядоченности кортежей и атрибутов

В каком бы порядке ни хранились данные в отношении, смысл их будет не изменен. Действительно, с помощью языков манипулирования данными при организации запросов всегда можно указать тот или иной порядок сортировки данных для результирующего набора данных.

Хотя некоторые реляционные СУБД позволяют обеспечить доступ к атрибуту отношения по порядковому номеру, который присваивается атрибуту в схеме отношения в этих СУБД, атрибуты в большинстве своем не упорядочены, и для ссылки на значение атрибута обычно (а языках манипулирования данными — обязательно) используется имя атрибута.

Обеспечение этих свойств позволяет, с одной стороны, СУБД хранить данные в произвольном порядке, с другой стороны, обеспечить разработчику и пользователю возможность манипулировать данными в отношениях без нарушения структурной целостности системы.

4.2.3. Атомарность значений атрибутов, первая нормальная форма

Одним из главных свойств отношений является соблюдение принципа нормализации, говоря по-другому, каждое отношение в БД должно удовлетворять первой нормальной форме (1-НФ).

Отношение находится в первой нормальной форме (нормализовано по 1-НФ) тогда и только тогда, когда значения его атрибутов являются атомарными, т.е. не содержат множества значений, иными словами значением атрибута отношения не может быть какое-либо отношение; значениями атрибутов не являются составные данные. Согласно [7], каждое отношение в 1-НФ является особым случаем ненормализованного отношения, но каждое ненормализованное отношение не находится в 1-НФ.

На рис. 4.2 приведен пример ненормализованного отношения ГРУППЫ. Можно сказать, что здесь мы имеем бинарное отношение, значением атрибута СТУДЕНТЫ которого является отношение.

№ группы	Студенты				
	№ зачетной книжки	ФИО	Пол	Место рождения	Дата рождения
412-1	1992412-11	Карасев А.А.	М	г. Чита	27.08.75
412-2	2002412-02	Красников И.И.	М	г. Бийск	12.02.83
432-1	1992432-11	Данилов О. В.	М	г. Алма-Ата	27.08.75
	1992432-12	Раевский А. И.	М	г. Бишкек	20.05.75
421-1	2002421-01	Иванков И.С.	Ж	г. Томск	11.10.85
	2002421-02	Авдеев Н.В	М	г. Омск	01.04.84

Рис. 4.2. Ненормализованное отношение ГРУППЫ

Отношение СТУДЕНТЫ (рис. 4.3) является нормализованным вариантом отношения ГРУППЫ. В этом отношении на пересечении столбца и строки находится только одно значение.

№ зачетной книжки	ФИО студента	Пол	Место Рождения	Дата Рождения	№ группы
1992412-11	Карасев А.А.	М	г. Чита	27.08.75	412-1
2002412-02	Красников И.И.	М	г. Бийск	12.02.83	412-2
1992432-11	Данилов О.В.	М	г. Алма-Ата	27.08.75	432-1
1992432-12	Раевский А.И.	М	г. Бишкек	20.05.75	432-1
2002421-01	Иванков И.С.	Ж	г. Томск	11.10.85	421-1
2002421-02	Авдеев Н.В	М	г. Омск	01.04.84	421-1

Рис. 4.3. Отношение СТУДЕНТЫ, находящееся в 1НФ

В современных реляционных СУБД, допускается хранить в полях таблиц сложные объекты (поля типа OLE), что, однако, не противоречит принципу атомарности данных, поскольку в данных полях содержится либо ссылка на внешний файл объекта, либо непосредственно сам OLE-объект, являющийся неделимой информационной единицей.

Что же касается составных данных, то возможность хранения в одном поле перечислимой информации типа «белый, синий, черный» остается на совести разработчика БД — либо принимается тезис о необходимости четкого описания объекта ПрО для обеспечения воз-

можности манипулировать информацией, представленной в этом поле, что влечет необходимость дальнейшей нормализации. Либо эта информация принимается как сопроводительная и имеет статус примечания.

4.2.4. Характеристика реляционной модели

Что касается реляционной модели данных, публикаций на эту тему в настоящее время достаточно много, однако фундаментальные понятия, термины и основы построения реляционной модели остаются неизменными на протяжении десятилетий. Так, например, основы проектирования реляционных моделей подробно изложены в книге Ш. Атре «Структурный подход к организации баз данных» еще в 1980 году.

К. Дейт в своей книге [10] дает следующее определение: ***реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной, манипуляционной и целостной.***

Единственной структурой данных, используемой в реляционных БД, является нормализованное n-арное отношение — это характеристика структурной части.

В *манипуляционной части* модели утверждаются два фундаментальных механизма манипулирования реляционными БД — реляционная алгебра и реляционное исчисление, на основе которых строятся все известные реляционные языки управления БД.

Что касается *целостной част*и реляционной модели данных, то для нее фиксируются два базовых требования, характерных для любой реляционной СУБД. Первое требование называется требованием целостности сущностей, согласно ему отношение должно обладать первичным ключом для обеспечения уникальности записей. Второе требование называется требованием целостности по ссылкам.

При описании сложных объектов ПрО, обойтись одним отношением, соблюдая принцип нормализации, бывает очень сложно, а зачастую и практически невозможно. Таким образом, один объект может быть описан в нескольких отношениях.

Поясним смысл требования целостности по сущностям на примере. Отобразим в реляционной БД сущность ФАКУЛЬТЕТ, содержащий информацию о группах – НОМЕР ГРУППЫ, КОЛИЧЕСТВО СТУДЕНТОВ — и о студентах факультета, а именно, № ЗАЧЕТНОЙ КНИЖКИ, ФИО СТУДЕНТА, ПОЛ, МЕСТО РОЖДЕНИЯ, ДАТА РОЖДЕНИЯ.

В результате проектирования БД получим два отношения СТУДЕНТЫ и ГРУППЫ, со схемами, представленными на рис. 4.4.

Отношение СТУДЕНТЫ

№ зачетной книжки	ФИО студента	Пол	Место рождения	Дата рождения	№ группы

РК — № ЗАЧЕТНОЙ КНИЖКИ

Отношение ГРУППЫ

№ группы	Количество студентов

РК — № ГРУППЫ

Рис. 4.4. Схемы отношений СТУДЕНТЫ и ГРУППЫ

Атрибут № ГРУППЫ появляется в отношении СТУДЕНТЫ для обеспечения возможности восстановить сущность ФАКУЛЬТЕТ. Значение атрибута НОМЕР ГРУППЫ отношения СТУДЕНТЫ должно соответствовать значению атрибута НОМЕР ГРУППЫ в каком-либо кортеже отношения ГРУППЫ. Атрибут № ГРУППЫ в отношении СТУДЕНТЫ называется ***внешним ключом***. Значения такого атрибута отношения однозначно характеризуют сущности, представленные кортежами другого отношения, — соответствуют значению его первичного ключа.

Таким образом, отношение, в котором на каком-либо атрибуте (может быть составном) определен внешний ключ, ссылается на отношение, в котором соответствующий атрибут является первичным ключом. Говорят также, что отношения связаны по некоторому ключу.

Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать) [1]. Для нашего примера это означает, что если для студента указан номер группы, то эта группа должна существовать.

Ограничения целостности сущностей и ограничения по ссылкам должны поддерживаться в большинстве современных СУБД.

Для обеспечения ограничения по ссылкам при добавлении и изменении данных ссылающегося отношения необходимо обеспечить проверку на ввод значений внешнего ключа. При изменении значения первичного ключа в отношениях необходимо изменять соответствующие значения внешних ключей. В некоторых СУБД эта процедура

производится автоматически, такой принцип получил название — каскадное обновление данных.

При удалении кортежа из отношения, на которое ведет ссылка, существуют три подхода, каждый из которых поддерживает целостность по ссылкам [1]. *Первый подход* заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). Во *втором подходе* при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным. Наконец, *третий подход* (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

При разработке структуры БД, необходимо обеспечить целостность данных либо на уровне СУБД либо на уровне прикладной программы.

Для реляционных СУБД, в которых поддерживаются домены, существует понятие целостности доменов. Обеспечение механизма целостности доменов гарантирует, что все значения некоторого атрибута принадлежат множеству допустимых значений [16]. Реализация механизма целостности доменов осуществляется с помощью предварительного задания характеристик домена в описательной части БД.

Соблюдение целостности БД является одним из условий обеспечения качественного хранения информации, а также гарантией отсутствия противоречивых данных, что позволяет постоянно актуализировать эти данные и использовать их многократно и без потерь.

4.2.5. Технология манипулирования данными в реляционной структуре

Согласно К. Дейту для манипуляционной составляющей определяются два базовых механизма манипулирования реляционными данными — реляционная алгебра (основана на теории множеств) и реляционное исчисление (основано на исчислении предикатов первого порядка), разделенное на два типа — исчисление доменов и исчисление предикатов.

Все современные языки манипулирования данными основаны на этих понятиях, а так как реляционная алгебра и реляционное исчисление замкнуты относительно понятия отношения, то любое выражение или формула могут быть представлены как отношения, что позволяет использовать их в других реляционных выражениях или формулах.

Применение реляционной алгебры и реляционного исчисления дает возможность интерпретировать сложные пользовательские запросы в виде простых предложений на языке манипулирования данными. По этой причине эти механизмы включены в реляционную модель данных.

Конкретный язык манипулирования реляционными БД называется реляционно полным, если любой запрос, выражаемый с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления, может быть выражен с помощью одного оператора этого языка [1].

В принципе для любого выражения реляционной алгебры можно построить формулу реляционного исчисления, приводящую к тому же результату, что и реляционное выражение и наоборот. Таким образом, эти два понятия являются эквивалентными. Для облегчения технической реализации пользовательских запросов в реляционной модели данных имеют место оба механизма манипулирования данными.

Контрольные вопросы

1. Перечислите основные понятия реляционной модели данных.
2. Сформулируйте основные свойства отношений.
3. Поясните смысл понятия целостности данных.
4. Дайте определение первой нормальной формы.

5. ОПЕРАЦИИ РЕЛЯЦИОННОЙ АЛГЕБРЫ И РЕЛЯЦИОННОЕ ИСЧИСЛЕНИЕ

5.1. Операции реляционной алгебры

5.1.1. Общий смысл операций реляционной алгебры

Остановимся более подробно на технологии манипулирования данными в реляционной модели, предложенной Коддом, для чего дадим определения операциям реляционной алгебры и технологии реляционного исчисления.

Поскольку отношения, как было определено в предыдущем разделе, есть множества, то для них справедливо применение теоретико-множественных операций, дополненных некоторыми специальными операциями, специфичными для баз данных.

Набор основных операций реляционной алгебры состоит из восьми операций, которые можно разделить на четыре традиционных теоретико-множественных операций и четыре специальные реляционные операции. Этот набор дополняет операция *присваивания*, сохраняющая в базе данных результаты вычисления, и операция *переименования атрибутов*, дающая возможность корректировать схему результирующего отношения.

Для теоретико-множественных операций характерны следующие операции:

прямое произведение отношений.
объединение отношений;
пересечение отношений;
взятие разности отношений;

В состав специальных реляционных операций принято включать операции:

ограничения отношения;
проекции отношения;
соединения отношений.
деление отношений.

Поскольку операции реляционной алгебры производятся над отношениями и результатом их выполнения является отношение, то говорят, что операции реляционной алгебры замкнуты относительно понятия отношения.

Прежде чем более подробно рассмотреть операции реляционной алгебры, дадим следующие общие определения.

Результатом выполнения прямого (декартового) произведения двух отношений-операндов является отношение, кортежи которого есть конкатенация (сцепление) кортежей первого и второго операндов.

Результатом выполнения операции объединения двух отношений является отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов.

В результате операции пересечения двух отношений производится отношение, включающее все кортежи, принадлежащие обоим отношениям-операндам.

В результате разности двух отношений создается отношение, включающее все кортежи, входящие в первое отношение-операнд, такие, что ни один из них не входит во второе отношение-операнд.

Результирующее отношение при проведении операции деления состоит из кортежей, включающих значения атрибутов кортежей первого операнда, таких, что множество значений оставшихся атрибутов (при фиксированном значении первых атрибутов) совпадает со множеством значений атрибутов второго операнда.

В результате операции ограничения отношения по некоторому условию производится отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.

В результате операции проекции отношения на фиксированный набор его атрибутов создается отношение, кортежи которого производятся путем взятия соответствующих значений из кортежей отношения-операнда с исключением атрибутов, не вошедших в первоначальный набор.

Результат соединения двух отношений по некоторому условию есть отношение, кортежи которого являются конкатенацией кортежей первого и второго отношений и удовлетворяют этому условию.

С помощью операции переименования атрибутов производится отношение, тело которого совпадает с телом операнда, но с заменой имен атрибутов.

С помощью операции присваивания можно сохранить результат вычисления реляционного выражения в отношении БД.

Для формирования пользовательских запросов с помощью операций реляционной алгебры можно строить вложенные выражения, содержащие последовательный набор таких операций.

5.1.2. Операция переименования

Иногда при выполнении той или иной операции реляционной алгебры производится отношение, содержащее атрибуты с некорректными именами, например, при выполнении декартового произведения

у отношений-операндов могут быть одноименные атрибуты, определенные на одном домене доменами. В результате такой операции было бы создано отношение с одноименными атрибутами, что противоречит определению схемы отношения как множества атрибутов, поскольку множество не должно содержать одинаковых элементов. Однако, удалив один из одноименных атрибутов, мы рискуем потерять часть информации. Для того чтобы избежать проблем с однозначным определением имен атрибутов, используется операция переименования. Для корректного выполнения реляционных операций, при возникновении конфликтов в именовании атрибутов, к одному из отношений-операндов необходимо сначала применить операцию переименования, а уже после этого выполнять основную операцию.

5.1.3. Особенности операций реляционной алгебры, операции объединения, пересечения и разности

Для парных теорико-множественных операций реляционной алгебры, необходимо выполнение некоторых условий.

Так, для операций объединения, пересечения и разности должно выполняться условие совместимости отношений по объединению. Если допустить в реляционной алгебре возможность теоретико-множественного объединения двух отношений с разными схемами, т.е. с разноименными атрибутами и доменами, то результатом выполнения этих операций будет не отношение реляционной модели, а таблица разнотипных и разноименных данных. Таким образом, операция объединения (пересечения и разности) над отношениями будет корректно выполняться в том и только в том случае, когда отношения обладают одинаковыми заголовками. Два отношения будут совместимы по объединению, если в заголовках отношений-операндов содержится один и тот же набор имен атрибутов, и одноименные атрибуты определены на одном и том же домене или, если понятие домена не поддерживается, одноименные атрибуты должны быть одного типа.

Если отношения-операнды совместимы во всем, кроме имен атрибутов, то для выполнения требования совместимости по объединению до выполнения основных операций необходимо применить операцию переименования для соответствующих атрибутов. Пример выполнения операции объединения представлен на рис. 5.1.

Если отношения-операнды совместимы по объединению, то при выполнении над ними операций объединения (пересечения и взятия разности) результатом является отношение со схемой, совпадающей со схемой каждого из отношений-операндов.

Отношение СТУДЕНТЫ 1-ГО КУРСА

№ студента	ФИО студента	Пол	Место рождения	Дата рождения	№ группы
1	Карасев А.А.	М	г. Чита	27.08.75	412-1
2	Данилов О. В.	М	г. Алма-Ата	27.08.75	432-1
3	Раевский А. И.	М	г. Бишкек	20.05.75	432-1

Отношение АБИТУРИЕНТЫ

№ абиту-риента	ФИО абитуриента	Пол	Место рождения	Дата рождения	№ группы
1001	Иванков И.С.	Ж	г. Томск	11.10.85	421-1
1002	Авдеев Н.В	М	г. Омск	01.04.84	421-1
1003	Красников И.И.		г. Бийск	12.02.83	412-2

Отношение СТУДЕНТЫ

№ студента	ФИО студента	Пол	Место рождения	Дата рождения	№ группы
1	Карасев А.А.	М	г. Чита	27.08.75	412-1
2	Данилов О. В.	М	г. Алма-Ата	27.08.75	432-1
3	Раевский А. И.	М	г. Бишкек	20.05.75	432-1
1001	Иванков И.С.	Ж	г. Томск	11.10.85	421-1
1002	Авдеев Н.В	М	г. Омск	01.04.84	421-1
1003	Красников И.И.	М	г. Бийск	12.02.83	412-2

Рис. 5.1. Пример операции объединения

Хотя любая из операций объединения, пересечения и разности может быть выражена через две другие, эти операции включены Кодом в манипуляционную часть реляционной модели с целью облегчения построения запросов к БД потенциальных пользователей СУБД.

5.1.4. Прямое (декартово) произведение

Прямым произведением отношений A и B со схемами, соответственно $(A_1, A_2, ..., A_n)$ и $(B_1, B_2, ..., B_m)$, является отношение C, со схемой

$(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$, которая равна объединению схем отношений A и B, кортежи отношения C получены в результате конкатенации (присоединения) каждого кортежа из отношения B с каждым кортежем отношения A.

Поясним смысл операции на примере (рис. 5.2).

Пусть имеем отношение ТУСУР**,** в котором содержится информация о факультетских командах по баскетболу ТУСУРа и отношение ТГУ, содержащее аналогичную информацию о командах ТГУ. Тогда декартовым произведением отношений ТУСУР и ТГУ будет отношение ИГРЫ**,** содержащее список участников, которые должны играть попарно.

ТУСУР

Факультет	Команда	Капитан
ФСУ	АОИ	Иванов
ФСУ	АСУ	Смирнов
РТФ	Радио	Комов

ТГУ

Факультет	Команда	Капитан
ФПМК	Инко	Сидоренко
ФПМК	КБ	Игумнов
МФУ	Управленец	Ткаченко

Игры

Факультет ТУСУР	Команда ТУСУР	Капитан ТУСУР	Факультет ТГУ	Команда ТГУ	Капитан ТГУ
ФСУ	АОИ	Иванов	ФПМК	Инко	Сидоренко
ФСУ	АОИ	Иванов	ФПМК	КБ	Игумнов
ФСУ	АОИ	Иванов	МФУ	Управленец	Ткаченко
ФСУ	АСУ	Смирнов	ФПМК	Инко	Сидоренко
ФСУ	АСУ	Смирнов	ФПМК	КБ	Игумнов
ФСУ	АСУ	Смирнов	МФУ	Управленец	Ткаченко
РТФ	Радио	Комов	ФПМК	Инко	Сидоренко
РТФ	Радио	Комов	ФПМК	КБ	Игумнов
РТФ	Радио	Комов	МФУ	Управленец	Ткаченко

Рис. 5.2. Пример операции прямого произведения

Два отношения *совместимы по взятию прямого произведения* тогда и только тогда, когда все атрибуты этих отношений имеют различные имена. Естественно, что два отношения можно сделать совместимыми по взятию прямого произведения с помощью операции переименования, примененной к одному из них.

Отметим, что вышеперечисленные операции являются ассоциативными, а именно: обозначим через OP любую из четырех операций, тогда (*R*1 OP *R*2) OP *R*3 = *R*1 (*R*2 OP *R*3), тогда справедлива следующая запись *R*1 OP *R*2 OP *R*3, где *R*1, *R*2 и *R*3 отношения, удовлетворяющие требованиям выполнения соответствующих операций. Кроме того, все операции, за исключением операции взятия разности, коммутативны, т.е. *R*1 OP *R*2 = *R*2 OP *R*1.

5.1.5. Специальные реляционные операции

Рассмотрим так называемые специальные операции реляционной алгебры, в состав которых входят операции ограничения, взятия проекции, соединения и деления.

Операция ограничения

Для выполнения операции ограничения (выборки) необходимо наличие двух операндов — ограничиваемого отношения и условия ограничения.

Условие ограничения может иметь либо вид (*a* comp-op *b*), где *a* и *b* — имена атрибутов ограничиваемого отношения, для которых осмысленна операция сравнения comp-op, либо вид (*a* comp-op const), где *a* — имя атрибута ограничиваемого отношения, а const — литерально заданная константа [1]. Операцию ограничения принято также называть операцией выбора.

Результатом операции ограничения является отношение с заголовком, совпадающим с заголовком отношения-операнда, в тело которого входят те кортежи отношения-операнда, для которых выполняется значением условие ограничения. Результат операция ограничения можно представить в виде горизонтальной вырезки кортежей отношения-операнда. На рис. 5.3 приведено отношение СТУДЕНТЫ (рис. 5.1), к которому применена операция ограничения с условием № группы = «421-1»

Отношение СТУДЕНТЫ

№ студента	ФИО студента	Пол	Место рождения	Дата рождения	№ группы
1001	Иванков И.С.	Ж	г. Томск	11.10.85	421-1
1002	Авдеев Н.В	М	г. Омск	01.04.84	421-1

Рис. 5.4. Результат выполнения операции ограничения

Рассмотрим механизм применение операции ограничении. Пусть UNION обозначает операцию объединения, INTERSECT — операцию

пересечения, а MINUS — операцию взятия разности. Для обозначения операции ограничения будем использовать конструкцию *A* WHERE comp, где *A* — ограничиваемое отношение, а comp — простое условие сравнения. Пусть comp1 и comp2 — два простых условия ограничения [1]. Тогда:

A WHERE comp1 AND comp2 обозначает то же самое, что и (*A* WHERE comp1) INTERSECT (*A* WHERE comp2)

A WHERE comp1 OR comp2 обозначает то же самое, что и (*A* WHERE comp1) UNION (A WHERE comp2)

A WHERE NOT comp1 обозначает то же самое, что и *A* MINUS (*A* WHERE comp1)

Таким образом, допускается использование операций ограничения, в которых условием ограничения является булевское выражение со стандартными логическими связками типа AND, OR, NOT и скобками.

Операция взятия проекции

Для выполнения операции взятия проекции требуется наличие проецируемого отношения *R* и списка имен атрибутов, входящих в схему отношения.

Результатом проекции отношения *R* по списку атрибутов *r*1, *r*2, ..., *rn* является отношение, с заголовком, определяемым множеством атрибутов *r*1, *r*2, ..., rn, и с телом, состоящим из кортежей вида <*r*1:*v*1, *r*2:*v*2, ..., *rn*:*vn*> таких, что в отношении *R* имеется кортеж, атрибут *r*1 которого имеет значение *v*1, атрибут *r*2 имеет значение *v*2, ..., атрибут *rn* имеет значение *vn*. Тем самым, при выполнении операции проекции выделяется «вертикальная» вырезка отношения-операнда [1]. Здесь следует заметить, что при появлении дублирующих кортежей, необходимо исключить таковые из результирующего набора. Так результатом выполнения операции взятия проекции отношения студены (рис. 5.1) на атрибуты № студента, ФИО студента, № группы является следующее отношение СТУДЕНТЫ_ГРУППА (рис. 5.4)

Отношение СТУДЕНТЫ_ГРУППА

№ студента	ФИО студента	№ группы
1	Карасев А.А.	412-1
2	Данилов О. В.	432-1
3	Раевский А. И.	432-1
1001	Иванков И.С.	421-1
1002	Авдеев Н.В	421-1
1003	Красников И.И.	412-2

Рис. 5.4. Результат операции взятия проекции

Операция соединения отношений

Результатом выполнения операции соединения отношения $R1$ по атрибуту X с отношением $R2$ по атрибуту Y является отношение R, множество всех кортежей которого является конкатенацией таких кортежей $a \in R1$ и кортежей $b \in R2$, для которых вычисляется условие $X * Y$, где $*$ есть операция сравнения типа « =, ≠, <, >, >=, <= ». Естественно, что X и Y должны быть определены на одном и том же домене.

Операцию соединения отношений обычно применяют для соединения двух и более таблиц в одну с целью получения сводных данных о некоторых связанных сущностях предметной области, либо, если одна сущность представлена несколькими отношениями, для восстановления таковой. На рис. 5.5 приведен пример использования операции соединения отношений *R1 (A1, A2) и R2 (A2, A3)* по совпадению значений атрибутов *A2* в обоих отношениях-операндах

R1

*A*1	*A*2
*a*1,1	*a*2,1
*a*1,2	*a*2,2
*a*1,3	*a*2,3
*a*1,4	*a*2,4

R2

A2	*A*3
*a*2,1	*a*3,1
*a*2,2	*a*3,2
*a*2,5	*a*3,3

R

*A*1	*A*2	*A*3
*a*1,1	*a*2,1	*a*3,1
*a*1,2	*a*2,2	*a*3,2

Рис. 5.5. Пример операции деления

Операция деления отношений

Рассмотрим два отношения $R(A1, A2)$ и $S(A2)$.

Кроме определения операции деления, данного в начале данного раздела справедливо следующее: деление R/S (рис. 5.3) представляет операция, определяющая такое наибольшее множество значение атрибута $A1$, что прямое произведение этого множества с отношением $S(A2)$ содержится в отношении R, т.е. для всех элементов отношения S, деление является операцией, определяющей такое *отношение*, что это отношение содержится в R.

ДЕЛИМОЕ (*R*)

*A*1	*A*2
*a*1,1	*a*2,1
*a*1,2	*a*2,2
*a*1,3	*a*2,3
*a*1,4	*a*2,4

.
.

ДЕЛИТЕЛЬ (*S*)

*A*2
*a*2,1
*a*2,2

РЕЗУЛЬТАТ

*A*1
*a*1,1
*a*1,2

Рис. 5.6. Пример операции деления

5.2. Реляционное исчисление

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка. Базисными понятиями исчисления являются понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы [1].

Пусть имеется БД, в состав которой входят два отношения СТУДЕНТЫ с заголовком (Stud_id, № ЗАЧЕТНОЙ КНИЖКИ, ФИО СТУ-ДЕНТА, МЕСТО РОЖДЕНИЯ, ДАТА РОЖДЕНИЯ, № ГРУППЫ) и отношение УСПЕВАЕМОСТЬ с заголовком (Stud_id, НАИМЕНО-ВАНИЕ ДИСЦИПЛИНЫ, СЕМЕСТР, ОЦЕНКА). Необходимо получит список студентов и наименований дисциплин, по которым получены неудовлетворительные оценки во втором семестре.

В результате формирования такого запроса с помощью операций реляционной алгебры, получилось бы алгебраическое выражение, которое читалось бы, следующим образом:

1) выполнить операцию соединения отношений СТУДЕНТЫ и УСПЕВАЕМОСТЬ по условию СТУДЕНТЫ.Stud_id = УСПЕВАЕ-МОСТЬ.Stud_id;

2) ограничить результирующее отношение по условию ОЦЕНКА = 2;

3) ограничить результирующее отношение по условию СЕМЕСТР = 2;

4) спроецировать полученное отношение на атрибуты ФИО СТУ-ДЕНТА, НАИМЕНОВАНИЕ ДИСЦИПЛИНЫ.

Таким образом, для получения результирующего отношения, удовлетворяющего нашим условиям, необходимо выполнить четыре последовательных шага, каждому из которых соответствует одна реляционная операция. Результатом формулирования этого же запроса с

помощью реляционного исчисления, явилась бы формула, которую можно было бы прочесть следующим образом:

Выбрать ФИО СТУДЕНТА и НАИМЕНОВАНИЕ ДИСЦИПЛИНЫ для студентов, таких, что существует кортеж в отношении Успеваемость с таким же значением Stud_id, что и в отношении СТУДЕНТЫ и значением СЕМЕСТР=2 и ОЦЕНКА=2.

Здесь мы указали характеристики результирующего набора кортежей, не указав способ его формирования. Для выполнения такого запроса СУБД сама обязана решить, какие операции по обработке данных и в каком порядке нужно произвести над отношениями СТУДЕНТЫ и УСПЕВАЕМОСТЬ.

Обычно говорят, что алгебраическая формулировка является процедурной, т.е. задающей правила выполнения запроса, а логическая — описательной (или декларативной), поскольку она всего лишь описывает свойства желаемого результата [1].

Реляционное исчисление условно делится на два типа исчислений: исчисление кортежей и исчисление доменов. В исчислении кортежей областью определения переменных являются отношения БД, другими словами, значение переменной есть кортеж отношения. В исчислении доменов соответственно областью определения переменных являются домены, на которых определяются соответствующие атрибуты отношений базы данных — допустимое значение переменной есть значение домена.

Для исчисления кортежей характерно такое понятие как правильно построенные формулы (WFF — Well-Formed Formula), предназначенные для выражения условий, накладываемых на кортежные переменные. В основе WFF лежат простые сравнения (comparison) значений атрибутов переменных или некоторых констант. Простое сравнение есть WFF, а WFF в круглых скобках есть простое сравнение. Примером простого сравнения может быть следующее предложение: «УСПЕВАЕМОСТЬ.ОЦЕНКА = 2».

Для построения более сложных WFF используются логические связки AND, OR, NOT, а также конструкция If ... Then.

Пусть F есть WFF, а C — простое сравнение, тогда правильно построенными формулами являются следующие выражения:

C AND F;

C OR F;

If C Then F.

Возможно также построение WFF с помощью кванторов существования и всеобщности.

Пусть F есть WFF, в которой используется некоторая переменная var, тогда следующие выражения есть WFF:

э var (F);

∀ var (F).

Переменные, входящие в правильно построенную формулу, могут быть связными или свободными. Переменная называется свободной, если она входит в состав такой WFF, при построении которой не использовались кванторы всеобщности или существования.

На практике это означает, что если для какого-то набора значений свободных кортежных переменных при вычислении WFF получено значение true, то эти значения кортежных переменных могут входить в результирующее отношение [1].

Переменная называется связанной, если при построении WFF ее имя использовано сразу после квантора э или ∀.

Связанная переменная не видна за пределами минимальной WFF, связавшей эту переменную, т.е. при вычислении значения такой WFF используется не одно значение связанной переменной, а ее полная область определения.

Говоря о свободных и связанных переменных, имеют в виду свободные и связанные вхождения переменных. Легко видеть, что если переменная var является связанной в WFF F, то во всех WFF, включающих данную, может использоваться имя переменной var, которая может быть свободной или связанной, но в любом случае не имеет никакого отношения к вхождению переменной var в WFF F [1]. В следующем примере мы имеем несколько разнотипных вхождений переменных СТУДЕНТЫ и УСПЕВАЕМОСТЬ.

э СТУДЕНТЫ (СТУДЕНТЫ.Stud_id=УСПЕВАЕМОСТЬ.Stud_id) AND ∀ УСПЕВАЕМОСТЬ (УСПЕВАЕМОСТЬ.СЕМЕСТР=2) AND ∀ УСПЕВАЕМОСТЬ (УСПЕВАЕМОСТЬ.ОЦЕНКА=2).

WFF предоставляют возможность сформулировать некоторые условия выборки данных из отношений БД. Для использования реляционного исчисления в реальной работе с БД необходимо применение еще одного компонента, определяющего набор и имена атрибутов результирующего отношения. Этот компонент реляционного исчисления называется целевым списком (target_list).

Что же касается исчисления кортежей, то выражением реляционного исчисления кортежей является конструкция вида target_list WHERE wff. Значением такого выражения будет отношение, тело которого определяется WFF, а набор атрибутов этого отношения определяется целевым списком.

В заключение раздела отметим, что на основе операций реляционной алгебры и реляционного исчисления строятся языки манипулирования данными, наиболее распространенным из которых является язык построения запросов QBE (Query by Example). В нем нашло широкое применение исчисление доменов и язык SQL, наиболее гибко сочетающий в себе операции реляционной алгебры и реляционное исчисление.

Контрольные вопросы

1. Перечислите и кратко охарактеризуйте операции реляционной алгебры.
2. Дайте определения совместимости отношений по объединению и по взятию прямого произведения.
3. Приведите примеры выполнения операций реляционной алгебры.
4. Поясните смысл применения реляционного исчисления

6. ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БД

6.1. Нормализация отношений

6.1.1. Термины и определения

При проектировании базы данных разработчику необходимо определить концептуальное и соответствующее ему физическое представление, после чего происходит создание внешних представлений с учетом потребностей прикладных программ, использующих спроектированную БД. При проектировании БД выделяют две стадии проектирования БД — логическое и физическое проектирование.

Логическое проектирование БД ставит цель представить реальную предметную область в абстрактных моделях так, чтобы эти модели данных максимально отражали в себе объекты выбранной предметной области. Что касается физического проектирования БД, то на этой стадии на основании спроектированной логической модели предметной области создается структура данных, определенная для конкретных СУБД, а также создание дополнительных элементов БД (триггеров, индексов и т.д.).

При проектировании реляционных БД трудно представить какие-либо общие решения по физическому проектированию. Ограничения в каждом конкретном случае накладывает СУБД, в которой предполагается создавать базу данных.

Таким образом, абстрагируясь от конкретных СУБД, будем считать, что при проектировании реляционной БД необходимо определить отношения, составляющие БД, их атрибуты, а также ключи и связи между отношениями.

Основной направляющей при проектировании реляционных БД является технология нормализации отношений, направленная на обеспечение безизбыточного и бесконфликтного хранения информации в отношениях БД. При обеспечении принципа нормализации процесс проектирования БД производится методом последовательных приближений к требуемому набору схем отношений. Нормализации подвергаются отношения, каждое из которых содержит характеристики объектов предметной области, при этом на каждом следующем шаге проектирования (нормализации) с помощью декомпозиции отношений, достигается такой набор схем отношений, что каждая следующая нормальная форма (НФ или NF) обладает лучшими свойствами, чем предыдущая.

При нормализации отношений каждой НФ соответствует свой набор ограничений, тогда отношение находится в какой-либо нормальной форме, если удовлетворяет характерному ей набору ограничений. Так, ограничением первой нормальной формы (1НФ), как мы отмечали в 4-м разделе, является условие атомарности атрибутов отношения. И поскольку в основе реляционной модели лежит отношение, находящееся в 1 НФ, в дальнейшем будем подразумевать, что все рассматриваемые нами отношения уже находятся в 1 НФ или, как говорят, нормализованы по 1НФ. Процесс нормализации позволяет разработчику БД глубже понять семантику атрибутов [7] при проектировании структуры БД и их взаимосвязи, а также облегчает проведение анализа предметной области.

За время развития технологии проектирования реляционных БД были выделены следующие нормальные формы:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Обычно на практике применение находят только первые три нормальные формы.

Для атрибутов отношений реляционных БД справедливо определение функциональной зависимости: так в отношении *R* атрибут *Y* функционально зависит от атрибута *X* в том и только в том случае, если каждому значению *X* соответствует одно значение *Y*. Схематично функциональную зависимость атрибута *Y* от атрибута *X* изображают так: *R.X -> R.Y* или *R*(*X*->*Y*). Из этого определения следуют определения полной и транзитивной функциональной зависимости. Функциональная зависимость атрибута *Y* от атрибута *X* называется полной, если атрибут *Y* не зависит функционально от любого точного подмножества *X* (следует учесть, что атрибуты *X* и *Y* могут быть составными). Функциональная зависимость *X -> Y* называется транзитивной, если имеется такой атрибут *Z*, что существуют функциональные зависимости *X -> Z* и *Z -> Y* и при этом отсутствует функциональная зависимость *Z -> X*. В отношении *R* атрибуты взаимно независимы, если ни один из этих атрибутов функционально не зависит от других.

6.1.2. Вторая нормальная форма

Рассмотрим процесс нормализации отношения по 2NF, т.е. приведем отношение (рис. 6.1), находящееся в первой нормальной форме ко второй нормальной форме.

Отношение УСПЕВАЕМОСТЬ

№ зачетной книжки	ФИО студента	Место Рождения	Дата Рождения	**Курс**	Средний балл
1992412-11	Карасев А.А.	г. Чита	27.08.75	1	4,5
1992412-11	Карасев А.А.	г. Чита	27.08.75	2	4,1
1992412-11	Карасев А.А.	г. Чита	27.08.75	3	5
1992412-11	Карасев А.А.	г. Чита	27.08.75	4	4,8
1992432-11	Данилов О. В.	г. Алма-Ата	27.08.75	1	4,6
1992432-11	Данилов О. В.	г. Алма-Ата	27.08.75	2	4,4
1992432-11	Данилов О. В.	г. Алма-Ата	27.08.75	3	4,2
1992432-11	Данилов О. В.	г. Алма-Ата	27.08.75	4	4,3
1992432-12	Раевский А. И.	г. Бишкек	20.05.75	1	3,8
. . .					
1992432-12	Раевский А. И.	г. Бишкек	20.05.75	4	5

Рис. 6.1. Отношение, не удовлетворяющее второй нормальной форме

Схема отношения УСПЕВАЕМОСТЬ: (№ ЗАЧЕТНОЙ КНИЖКИ, ФИО СТУДЕНТА, МЕСТО РОЖДЕНИЯ, ДАТА РОЖДЕНИЯ, КУРС, СРЕДНИЙ БАЛЛ). Первичным ключом отношения является совокупность атрибутов № ЗАЧЕТНОЙ КНИЖКИ, КУРС.

Можно выявить следующие функциональные зависимости:

№ ЗАЧЕТНОЙ КНИЖКИ -> ФИО СТУДЕНТА

№ ЗАЧЕТНОЙ КНИЖКИ -> МЕСТО РОЖДЕНИЯ

№ ЗАЧЕТНОЙ КНИЖКИ -> ДАТА РОЖДЕНИЯ

№ ЗАЧЕТНОЙ КНИЖКИ, КУРС -> СРЕДНИЙ БАЛЛ

Первичный ключ данного отношения состоит из атрибутов № ЗАЧЕТНОЙ КНИЖКИ, КУРС, однако в нашем случае, атрибуты ФИО СТУДЕНТА, МЕСТО РОЖДЕНИЯ и ДАТА РОЖДЕНИЯ функционально зависят только от части первичного ключа — № ЗАЧЕТНОЙ КНИЖКИ. Таким образом, при работе с таким ненормализованным отношением не возможно обеспечить корректную работу при выполнении операции вставки нового кортежа, при занесении данных о студентах, сведений об успеваемости которых еще нет, т.к. первичный ключ не может содержать неопределенное значение (в нашем случае

часть ключа КУРС не определена). При удалении записи из отношения мы теряем связь конкретного студента с его успеваемостью за конкретный курс. Аналогично мы получим неверный результат при выполнении подсчета общего количества студентов. Такие неприятные явления называются аномалиями схемы отношения или коллизиями. Эти недостатки реляционных отношений устраняются путем нормализации по 2 НФ.

Отношение R находится во второй нормальной форме (2НФ) тогда и только тогда, когда отношение находится в первой нормальной форме, и каждый его не ключевой атрибут полностью зависит от первичного ключа. Или, что тоже справедливо, отношение, находящееся во второй нормальной форме не содержит атрибутов, зависящих от части ключа.

Приведение данного отношения к 2НФ заключается в разбиении (декомпозиции) на два отношения, удовлетворяющих соответствующим требованиям нормализации. Можно произвести следующую декомпозицию отношения УСПЕВАЕМОСТЬ в два отношения СТУДЕНТЫ и УСПЕВАЕМОСТЬ СТУДЕНТОВ (рис. 6.2).

СТУДЕНТЫ

№ зачетной книжки	ФИО студента	Место Рождения	Дата Рождения
1992412-11	Карасев А.А.	г. Чита	27.08.75
1992432-11	Данилов О. В.	г. Алма-Ата	27.08.75
1992432-12	Раевский А. И.	г. Бишкек	20.05.75

УСПЕВАЕМОСТЬ СТУДЕНТОВ

№ зачетной книжки	**Курс**	Средний балл
1992412-11	1	4,5
1992412-11	2	4,1
1992412-11	3	5
1992412-11	4	4,8
1992432-11	1	4,6
1992432-11	2	4,4
1992432-11	3	4,2
1992432-11	4	4,3
1992432-12	1	3,8
1992432-12	4	5

Рис. 6.2. Результат декомпозиции отношения УСПЕВАЕМОСТЬ

Первичным ключом отношения СТУДЕНТЫ являются атрибут № ЗАЧЕТНОЙ КНИЖКИ.

Можно выявить следующие функциональные зависимости:

№ ЗАЧЕТНОЙ КНИЖКИ -> ФИО СТУДЕНТА;

№ ЗАЧЕТНОЙ КНИЖКИ -> МЕСТО РОЖДЕНИЯ;

№ ЗАЧЕТНОЙ КНИЖКИ -> ДАТА РОЖДЕНИЯ;

Первичным ключом отношения УСПЕВАЕМОСТЬ СТУДЕНТОВ является атрибуты № ЗАЧЕТНОЙ КНИЖКИ, КУРС.

В этом отношении существует одна функциональная зависимость:

№ ЗАЧЕТНОЙ КНИЖКИ, КУРС -> СРЕДНИЙ БАЛЛ

Каждое из этих двух отношений находится в 2НФ, и в них устранены отмеченные выше аномалии (легко проверить, что все указанные операции выполняются без проблем).

В отношении помимо одного первичного ключа могут находиться атрибуты, по значению которых также можно однозначно определить записи, такие атрибуты называются альтернативными ключами. Если допустить наличие нескольких ключей, то определение 2НФ примет следующий вид: *отношение R находится во второй нормальной форме (2НФ) в том и только в том случае, когда оно находится в 1NF, и каждый его неключевой атрибут полностью зависит от каждого ключа этого отношения.*

6.1.3. Третья нормальная форма

Добавим в отношение СТУДЕНТЫ два атрибута: № ГРУППЫ и ФИО КУРАТОРА (рис. 6.3).

№ зачетной книжки	ФИО студента	Дата рождения	Место рождения	№ Группы	ФИО куратора
1992412-11	Карасев А.А.	27.08.75	г. Чита	412-1	Самойлов С.С.
1992432-11	Данилов О. В.	27.08.75	г. Алма-Ата	432-1	Авдеев Р.М
1992432-12	Раевский А. И.	20.05.75	г. Бишкек	432-1	Авдеев Р.М
. . .					
1992432-22	Глазов О.А	04.07.75	г. Киров	432-1	Авдеев Р.М

Рис. 6.3. Отношение, не удовлетворяющее третьей нормальной форме

Первичным ключом отношения СТУДЕНТЫ является атрибут № ЗАЧЕТНОЙ КНИЖКИ. Отношение находится во второй нормальной форме, поскольку отсутствуют зависимости атрибутов от части первичного ключа.

Функциональная зависимость № ЗАЧЕТНОЙ КНИЖКИ -> ФИО КУРАТОРА транзитивная, поскольку является следствием функциональных зависимостей № ЗАЧЕТНОЙ КНИЖКИ -> № ГРУППЫ и № ГРУППЫ -> ФИО КУРАТОРА. Другими словами, ФИО КУРАТОРА является характеристикой не студента, а группы, в которой он проходит обучение.

Легко заметить, что в рассматриваемом отношении существуют следующие аномалии включения и удаления. В результате выполнения операции включения мы не сможем занести в базу данных информацию о кураторе группы до тех пор, пока в этой группе не появится хотя бы один студент, поскольку первичный ключ не может содержать неопределенное значение, однако на практике еще до составления списков групп уже известны их кураторы. При удалении кортежа, описывающего последнего студента данной группы, мы рискуем потерять информацию о том, кто из преподавателей кафедры, на которой обучается студент, являлся куратором этой группы.

Чтобы корректно изменить ФИО куратора группы, необходимо последовательно изменить все кортежи, описывающие студентов этой группы, т.е. в отношении СТУДЕНТЫ, благодаря добавлению двух новых атрибутов, по-прежнему существуют аномалии. Их можно устранить путем дальнейшей нормализации отношения. Дадим определение третьей нормальной формы

Отношение R находится в третьей нормальной форме (3НФ) в том и только в том случае, если находится в 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа, т.е. среди атрибутов отношения нет атрибутов, транзитивно зависящих от ключа (среди его неключевых атрибутов нет зависящих от другого неключевого атрибута).

Приведение отношения к 3НФ (нормализация по 3НФ) также заключается в декомпозиции этого отношения. Можно произвести декомпозицию отношения СТУДЕНТЫ в два отношения СПРАВОЧНИК СТУДЕНТОВ и ГРУППЫ. Результат нормализации представлен на (рис. 6.4).

СПРАВОЧНИК СТУДЕНТОВ

№ зачетной книжки	ФИО студента	Дата рождения	Место рождения	№ Группы
1992412-11	Карасев А.А.	27.08.75	г. Чита	412-1
1992432-11	Данилов О. В.	27.08.75	г. Алма-Ата	432-1
1992432-12	Раевский А. И.	20.05.75	г. Бишкек	432-1
1992432-22	Глазов О.А	04.07.75	г. Киров	432-1

КУРАТОРЫ ГРУПП

№ Группы	ФИО куратора
412-1	Самойлов С.С.
432-1	Авдеев Р.М

Рис. 6.4. Результат декомпозиции отношения СТУДЕНТЫ

Первичным ключом отношения СПРАВОЧНИК СТУДЕНТОВ является атрибут № ЗАЧЕТНОЙ КНИЖКИ. В отношении имеют место следующие функциональные зависимости:

№ ЗАЧЕТНОЙ КНИЖКИ -> ФИО СТУДЕНТА;

№ ЗАЧЕТНОЙ КНИЖКИ -> ДАТА РОЖДЕНИЯ;

№ ЗАЧЕТНОЙ КНИЖКИ -> МЕСТО РОЖДЕНИЯ;

№ ЗАЧЕТНОЙ КНИЖКИ -> № ГРУППЫ.

Первичным ключом отношения КУРАТОРЫ ГРУПП является атрибут № ГРУППЫ. Для этого отношения характерна одна функциональная зависимость: № ГРУППЫ -> ФИО КУРАТОРА.

Результирующие отношения находятся в 3НФ и свободны от отмеченных аномалий.

При проектировании БД третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

6.1.4. Нормальная форма Бойса-Кодда

Нормальная форма Бойса-Кодда (BCNF) является расширением 3НФ, вводит дополнительное ограничение по сравнению с 3НФ и является промежуточным звеном между 3НФ и 4НФ. Нормальную форму Бойса-Кодда принято также называть *усиленной 3-й нормальной формой*

Рассмотрим отношение УСПЕВАЕМОСТЬ ПО ДИСЦИПЛИНАМ, в котором представлена информация об успеваемости студентов,

обучающихся по индивидуальному плану, т.е. набор предметов для студентов одной группы отличается и зависит от студента (рис. 6.5):

УСПЕВАЕМОСТЬ ПО ДИСЦИПЛИНАМ

№ зачетной книжки	**ФИО студента**	**Код дисциплины**	Рейтинг
1992412-11	Карасев А.А.	1	100
1992412-11	Карасев А.А.	2	90
1992412-11	Карасев А.А.	3	98
1992432-22	Глазов О.А	1	111
1992432-22	Глазов О.А	4	122
.			
1992432-12	Раевский А. И.	1	101
1992432-12	Раевский А. И.	2	120
1992432-12	Раевский А. И.	3	95

Рис. 6.5. Отношение, не удовлетворяющее BCNF

В данном отношении могут быть определены два первичных ключа, состоящие из следующих атрибутов:

PK1: № ЗАЧЕТНОЙ КНИЖКИ, КОД ДИСЦИПЛИНЫ

PK2: ФИО СТУДЕНТА, КОД ДИСЦИПЛИНЫ

Тогда в отношении будут существовать следующие функциональные зависимости:

№ ЗАЧЕТНОЙ КНИЖКИ -> ФИО СТУДЕНТА;

№ ЗАЧЕТНОЙ КНИЖКИ -> КОД ДИСЦИПЛИНЫ;

ФИО СТУДЕНТА -> КОД СТУДЕНТА;

ФИО СТУДЕНТА -> КОД ДИСЦИПЛИНЫ;

№ ЗАЧЕТНОЙ КНИЖКИ, КОД ДИСЦИПЛИНЫ -> РЕЙТИНГ;

ФИО СТУДЕНТА, КОД ДИСЦИПЛИНЫ -> РЕЙТИНГ.

Ситуация, когда отношение будет находиться в 3NF, но не в BCNF, возникает при условии, что отношение имеет несколько возможных первичных ключей, состоящих из нескольких атрибутов и имеющих общий атрибут. Для всех прочих отношений 3NF и BCNF эквивалентны.

Отношение находится в BCNF, если оно находится в 3НФ и в нем отсутствуют зависимости атрибутов, входящих в первичный ключ от неключевых атрибутов.

Очевидно, что это требование не выполнено для нашего отношения. Можно произвести его декомпозицию к отношениям СТУДЕНТЫ и УСПЕВАЕМОСТЬ (рис. 6.6).

СТУДЕНТЫ

№ зачетной книжки	**ФИО студента**
1992412-11	Карасев А.А.
1992432-22	Глазов О.А
1992432-12	Раевский А. И.

УСПЕВАЕМОСТЬ

№ зачетной книжки	**Код дисциплины**	Рейтинг
1992412-11	1	100
1992412-11	2	90
1992412-11	3	98
1992432-22	1	111
1992432-22	4	122
1992432-12	1	101
1992432-12	2	120
1992432-12	3	95

Рис. 6.6. Отношения, находящиеся в BCNF

Тогда, возможные ключи отношения СТУДЕНТЫ, № ЗАЧЕТНОЙ КНИЖКИ либо ФИО СТУДЕНТА. Функциональные зависимости в этом отношении:

№ ЗАЧЕТНОЙ КНИЖКИ -> ФИО СТУДЕНТА,

ФИО СТУДЕНТА -> № ЗАЧЕТНОЙ КНИЖКИ

Возможный ключ отношения УСПЕВАЕМОСТЬ — № ЗАЧЕТНОЙ КНИЖКИ, КОД ДИСЦИПЛИНЫ. Функциональная зависимость:

№ ЗАЧЕТНОЙ КНИЖКИ, КОД ДИСЦИПЛИНЫ -> РЕЙТИНГ.

6.1.5. Четвертая нормальная форма

Четвертая нормальная форма (4НФ) касается отношений, для которых характерно наличие повторяющихся наборов данных. В этом случае декомпозиция, основанная на функциональных зависимостях, не приводит к исключению такой избыточности. В этом случае используют декомпозицию, основанную на многозначных зависимостях.

В отношении *R* (*A, B, C*) существует многозначная зависимость *A* ->-> *B* в том и только в том случае, если множество значений *B,* соответствующее паре значений *A* и *C*, зависит только от *A* и не зависит от *C* [1].

Можно дать еще одно определение многозначной функциональной зависимости [10]: пусть имеется отношение *R*, с атрибутами *A, B, C*; атри-буты *B,C* многозначно зависят от *A* (*A->->B|C*) тогда и только тогда, когда из того, что в отношении содержатся кортежи *r*1 = (*a, b, c*1) и *r*2 = (*a, b*1, *c*) следует, что в отношении содержится также и кортеж *r*3 = (*a,b, c*).

Рассмотрим пример следующего отношения ИЗУЧАЕМЫЕ ДИСЦИПЛИНЫ (рис. 6.7).

Отношение содержит код специальности, для каждой специальности имеется перечень предметов, а также список литературы, рекомендуемый при изучении соответствующего курса. Студенты специальности могут изучать несколько предметов, и по разным предметам может быть рекомендована одинаковая литература.

ИЗУЧАЕМЫЕ ДИСЦИПЛИНЫ

Предмет	Код Специальности	Литература
ОБД	2202	Ш. Атре
ОБД	2202	Алан Р. Саймон
ПОВС	2202	Алан Р. Саймон
ПОВС	2202	Ш. Атре
ОБД	2204	Ш. Атре
ОБД	2204	Алан Р. Саймон

Рис. 6.7. Отношение, содержащее многозначные зависимости

Каждый кортеж отношения связывает некоторый предмет с конкретной специальностью, на которой изучается данный предмет, и литературой, рекомендованной при изучении данного предмета. Отсюда видно, что единственным ключом, возможным в нашем отношении, является составной атрибут КОД СПЕЦИАЛЬНОСТИ, ПРЕДМЕТ, ЛИТЕРАТУРА.

Следовательно, отношение ИЗУЧАЕМЫЕ ДИСЦИПЛИНЫ находится в BCNF. Но при этом оно обладает недостатками: если указать, что предмет изучается еще на одной специальности, то необходимо добавить в отношение весь перечень книг рекомендованных для прочтения по данной тематике:

ПОВС	2204	Алан Р. Саймон
ПОВС	2204	Ш. Атре

Так как наше отношение не содержит никаких функциональных зависимостей, то декомпозиция отношения не может быть выполнена на основе функциональных зависимостей. Однако заметно, что какая-то взаимосвязь между атрибутами имеется. Эта взаимосвязь и есть многозначная зависимость.

В отношении ИЗУЧАЕМЫЕ ДИСЦИПЛИНЫ существуют две многозначные зависимости:

ПРЕДМЕТ ->>КОД СПЕЦИАЛЬНОСТИ — зависимость множества значений атрибута КОД СПЕЦИАЛЬНОСТИ от множества значений атрибута ПРЕДМЕТ;

ПРЕДМЕТ ->> ЛИТЕРАТУРА — зависимость множества значений атрибута ЛИТЕРАТУРА от множества значений атрибута ПРЕДМЕТ.

В общем случае в отношении *R* (*A, B, C*) существует многозначная зависимость *A*->> *B* в том и только в том случае, когда существует многозначная зависимость *A* ->> *C* [1].

Многозначная зависимость *A*->->*B*|*C* называется нетривиальной, если не существует функциональных зависимостей *A*->*B* и *A*->*C*.

В нашем примере мы имеем дело именно с нетривиальной многозначной зависимостью

Нормализация отношений по 4НФ основывается на следующей теореме [1].

Теорема Фейджина

Отношение *R* (*A, B, C*) можно спроецировать без потерь в отношения *R*1 (*A, B*) и *R*2 (*A, C*) в том и только в том случае, когда существует MVD *A* -> -> *B* | *C*.

Проецирование без потерь — это такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений [1].

Отношение *R* находится в 4НФ в том и только в том случае, если в случае существования многозначной зависимости *A* ->> *B* все остальные атрибуты *R* функционально зависят от *A* [1].

Или иначе, отношение находится в четвертой нормальной форме тогда и только тогда, когда отношение находится в BCNF и не содержит нетривиальных многозначных зависимостей.

В нашем примере можно произвести декомпозицию отношения ИЗУЧАЕМЫЕ ДИСЦИПЛИНЫ в два отношения ПРЕДМЕТЫ-СПЕЦИ-АЛЬНОСТЬ и ПРЕДМЕТЫ-ЛИТЕРАТУРА:

ПРЕДМЕТЫ-СПЕЦИАЛЬНОСТЬ (ПРЕДМЕТ, КОД СПЕЦИАЛЬНОСТИ);

ПРЕДМЕТЫ-ЛИТЕРАТУРА (ПРЕДМЕТ, ЛИТЕРАТУРА).

Оба эти отношения находятся в 4НФ и свободны от отмеченных аномалий. Таким образом, полученные отношения остались полностью ключевыми, и в них по-прежнему нет функциональных зависимостей.

Отношения с нетривиальными многозначными зависимостями возникают, как правило, в результате естественного соединения двух отношений по общему полю, которое не является ключевым ни в од-

ном из отношений. Фактически это приводит к попытке хранить в одном отношении информацию о двух независимых сущностях. *В качестве еще одного примера можно привести ситуацию, когда сотрудник может иметь много работ и много детей. Хранение информации о работах и детях в одном отношении приводит к возникновению нетривиальной многозначной зависимости Работник->->Работа|Дети.*

6.1.6. Пятая нормальная форма

В предыдущих примерах процесс нормализации строился путем декомпозиции одного отношения в два. В отдельных случаях это сделать не удается, тогда приходится проводить декомпозицию в большее число отношений, каждое из которых обладает лучшими свойствами.

Рассмотрим отношение (рис. 6.8).

СПЕЦИАЛЬНОСТИ СТУДЕНТОВ

Код студента	№ группы	Код специальности
1	422-1	2202
1	422-1	2204
1	472-1-В	2208
1	472-1-В	2210
2	422-1	2202

Рис. 6.8. Пример отношения, не находящегося в 5НФ

Предположим, что один и тот же студент может обучаться в нескольких группах и обучаться в каждой группе по нескольким специальностям. Первичным ключом этого отношения является полная совокупность его атрибутов, отсутствуют функциональные и многозначные зависимости.

Поэтому отношение находится в 4НФ. Однако в нем могут существовать аномалии, которые нельзя устранить путем декомпозиции этого отношения в два. Всевозможные проекции отношения, включающие по два атрибута, представлены на рис. 6.9.

Проекция $R1 = R[A, B]$

Код студента	№ группы

1	422-1
1	472-1-В
2	422-1

Проекция $R2 = R[A, C]$

Код студента	Код специальности
1	2202
1	2204
1	2208
1	2210
2	2202

Проекция $R2=R[B,C]$

№ группы	Код Специальности
422-1	2202
422-1	2204
472-1-В	2208
472-1-В	2210

Рис. 6.9. Проекции отношения СПЕЦИАЛЬНОСТИ СТУДЕНТОВ

Из этого примера нетрудно заметить, что отношение *R* не восстанавливается ни по одному из попарных соединений *R*1и *R*2, *R*2 и *R*3 и т.д. Например, в результате соединения отношения *R*1 и *R*2 получим отношение *R*4, которое отличается от первоначального отношения *R* (рис. 6.10).

Отношение *R*4

Код студента	№ группы	Код специальности
1	422-1	2202
1	422-1	2204
1	422-1	2208
1	422-1	2210
1	472-1-В	2202
1	472-1-В	2204
1	472-1-В	2208
1	472-1-В	2210
2	422-1	2202

Рис. 6.10. «Восстановленное» отношение СПЕЦИАЛЬНОСТИ СТУДЕНТОВ

Аналогично можно попытаться произвести другие попарные соединения, и ни одно из них не восстанавливает отношение *R*, однако

первоначальный вид нашего отношения *R* восстанавливается соединением всех трех проекций. Это говорит о том, что между атрибутами этого отношения имеется некоторая зависимость, хотя она и не является ни функциональной, ни многозначной. Существующие в данном отношении зависимости называются зависимостями соединения.

Отношение *R* (*X, Y, ..., Z*) удовлетворяет зависимости соединения * (*X, Y, ..., Z*) в том и только в том случае, когда *R* восстанавливается без потерь путем соединения своих проекций на *X, Y, ..., Z*. (*X, Y, …, Z*) могут быть составными атрибутами) [1]. Зависимость соединения * (*X, Y, ..., Z*) называется тривиальной зависимостью соединения, если выполняется одно из условий:

1) либо все множества атрибутов (*X, Y, ..., Z*) содержат потенциальный ключ отношения *R*;

2) либо одно из множеств атрибутов совпадает со всем множеством атрибутов отношения *R*.

Тогда получаем следующее определение: отношение *R* находится в пятой нормальной форме (нормальной форме проекции-соединения PJ/NF) в том и только в том случае, когда любая зависимость соединения в *R* следует из существования некоторого возможного ключа в *R* [1]. Существует несколько более простых для понимания определений 5НФ:

1) отношение находится в 5НФ тогда и только тогда, когда любая зависимость по соединению в нем определяется только его возможными ключами;

2) отношение *R* находится в пятой нормальной форме (5НФ) тогда и только тогда, когда любая имеющаяся зависимость соединения является тривиальной.

Введем следующие имена составных атрибутов:

СГ = {КОД СТУДЕНТА, № ГРУППЫ};

СС = {КОД СТУДЕНТА, КОД СПЕЦИАЛЬНОСТИ};

ГС = {№ ГРУППЫ, КОД СПЕЦИАЛЬНОСТИ}.

Предположим, что в отношении СТУДЕНТЫ существует зависимость соединения:

* (СГ, СС, ГС), при определенных ограничениях предметной области, эта зависимость нетривиальна, проведем декомпозицию отношения. Декомпозиции нашего отношения в три новых отношения будет выглядеть следующим образом:

СТУДЕНТЫ-ГРУППЫ (КОД СТУДЕНТА, № ГРУППЫ);

СТУДЕНТЫ-СПЕЦИАЛЬНОСТИ (КОД СТУДЕНТА, КОД СПЕЦИАЛЬНОСТИ);

ГРУППЫ-СПЕЦИАЛЬНОСТИ (№ ГРУППЫ, КОД СПЕЦИАЛЬНОСТИ).

Получившиеся отношения свободны от нетривиальных зависимостей соединения и находятся в 5НФ.

5НФ — это последняя нормальная форма, которую можно получить путем декомпозиции отношения. На практике 5NF практически не используется.

6.2. Моделирование данных с помощью ER-диаграмм. Семантическое моделирование

6.2.1. Основные понятия модели Сущность-Связь

При проектировании информационных систем иногда трудно моделировать предметную область на основе таблиц. Реляционная модель в полной мере не дает представления семантики предметной области.

Потребности проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области вызвали к жизни новое направление — проектирование семантических моделей данных. Главным назначением семантических моделей является обеспечение возможности выражения семантики данных [1].

Проектировщик информационной системы на первой стадии проектирования БД зачастую использует семантическое моделирование. На этом этапе в идеологии семантической модели производится концептуальная схема базы данных, которая затем преобразуется к реляционной схеме. Этот процесс может выполняться как вручную, так и с помощью различных программных средств проектирования моделей данных. В результате этих преобразований на базе семантической модели производится реляционная схема базы данных в третьей нормальной форме с учетом специфики конкретной СУБД.

Рассмотрим одну из разновидностей семантического моделирования, получившей в последние годы большую популярность среди проектировщиков БД, — модель «Сущность-Связь» или ER-модель (Entity-Relationship-model).

Методология ER-модели была разработана Ченом (Chen) в середине 70-х годов XX века и получила дальнейшее развитие в работах Баркера. Идея моделирования предметной области заключается в возможности использования графических диаграмм, включающих необходимые для описания предметной области сущности и связи между ними. Благодаря простоте восприятия и наглядности представления концептуальных схем БД, ER-модели нашли широкое применение в CASE-системах, обеспечивающих автоматизированное проектирование БД. Необходимо отметить, что нотация может несколько отли-

чаться в зависимости от используемого разработчиком средства моделирования. Однако термины и элементы, которыми оперируют при разработке модели, присутствуют всегда и несут идентичную смысловую нагрузку.

В ER-модели используются понятия сущности предметной области, связей между ними и атрибутов сущностей.

Сущность — это объект предметной области, элемент информационной системы. В ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности и ее атрибуты.

Атрибут сущности — простейший элемент информационной системы, формирующийся вручную или посредством классификаторов из элементов первичной однородной фактографической информации, разделенной по назначению

Атрибут может быть либо обязательным, либо необязательным. Обязательность атрибута означает, что атрибут не может принимать неопределенных значений (null values). Атрибут может быть либо описательным (т.е. обычным дескриптором сущности), либо входить в состав уникального идентификатора (первичного ключа).

В общем виде при описании модели имена атрибутов могут быть опущены, а для описания сущности могут быть добавлены примеры объектов. Ниже изображена сущность ВУЗ с набором атрибутов и объектами ТУСУР и ТГУ:

Рис. 6.11. Пример графического изображения сущности ВУЗ

При создании ER-модели необходимо учитывать требование уникальности экземпляров сущности, то есть, каждая сущность должна обладать набором атрибутов, однозначно характеризующим экземпляр сущности.

Линии, связывающие сущности в ER-моделях, определяют взаимосвязи между объектами предметной области. Связь — это ассоциация между сущностями, где, как правило, каждый экземпляр одной сущности (родительская сущность) ассоциируется с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком, а каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя.

Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя. Связи в ER-моделях могут определяться как между двумя разными сущностями, так и между сущностью и ей же самой, такая связь называется рекурсивной связью.

Связи в ER-диаграммах могут быть типа один к одному (1:1), один ко многим (1:M), многие ко многим (M:M).

В месте стыковки связи с сущностью используются трехточечный вход, если для этой сущности в связи могут использоваться много (many) экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный — прерывистой линией [1]. На концах связи указывается имя конца связи, обязательность связи, при необходимости также указывается степень конца связи, то есть количество связываемых экземпляров данной сущности.

На рис. 6.12 представлен фрагмент ER-модели предметной области ВУЗ. Здесь показана связь между сущностями СТУДЕНТЫ и ЗАДОЛЖЕННОСТЬ ЗА ОБУЧЕНИЕ, которая связывает ЗАДОЛЖЕННОСТИ и СТУДЕНТЫ. Конец связи с именем ДЛЯ позволяет связывать с одним студентом несколько задолженностей, причем каждый экземпляр сущности ЗАДОЛЖЕННОСТЬ ЗА ОБУЧЕНИЕ должен быть связан хотя бы с одним студентом (Конец связи с именем ИМЕЕТ), при этом не каждый студент может иметь задолженность за обучение. Другими словами, каждый студент может иметь задолженности за обучение, и каждая задолженность характерна только для одного студента.

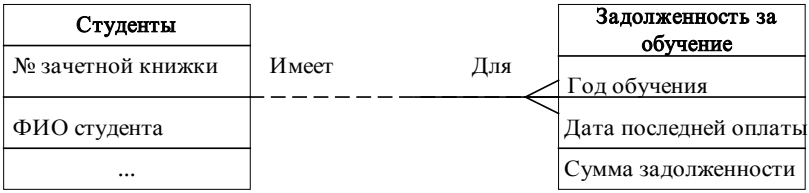

Рис. 6.12. Пример связи между сущностями ER-модели

На рис. 6.13 представлен пример рекурсивной связи, связывающей сущность СОТРУДНИК с ней же самой.

Рис. 6.13. Пример рекурсивной связи

Конец связи с именем ПОДЧИНЕННЫЙ говорит о том, что у одного сотрудника может быть более одного подчиненного, при этом конец связи с именем НАЧАЛЬНИК означает, что не каждый сотрудник может иметь подчиненных, т.е. быть начальником может не каждый сотрудник. Другими словами каждый сотрудник может быть начальником и являться подчиненным и при этом сам являться подчиненным только одного сотрудника.

6.2.2. Принцип нормализации ER-схем

Нормальные формы ER-схем имеют много общего с нормализацией реляционных отношений.

Для первой нормальной формы ER-схемы характерно отсутствие повторяющихся атрибутов — производится выявление так называемых неявных сущностей и образование на их основе новых сущностей.

Для второй нормальной формы характерно отсутствие атрибутов, зависящих от части уникального идентификатора сущности. На основе этой части уникального идентификатора выделяется отдельная сущность.

Для обеспечения третьей нормальной формы необходимо устранить атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. На основе этих атрибутов также строятся новые сущности.

Нормальные формы более высоких порядков в ER-моделях не находят практического применения.

6.2.3. Другие элементы ER-модели

В ряде случаев при проектировании модели предметной области основных понятий ER-модели недостаточно, тогда приходится исполь-

зовать расширенные возможности технологии создания ER-моделей, среди них можно выделить следующие конструкции:

1) *домены*. Под термином домен в данном случае следует понимать поименованный набор правил, которые являются общими для атрибутов, на которые распространяется действие этого домена. Преимущества использования доменов очевидны: определив правило один раз, можно добавлять его к атрибутам различных сущностей с целью стандартизации их характеристик;

2) *супертипы сущностей*. Этот элемент модели присутствует не во всех средствах проектирования ER диаграмм, однако наличие его обеспечивает возможность наследования типа сущности, исходя из одного или нескольких так называемых супертипов;

3) *подтипы сущностей*. Любая сущность ER-диаграммы может быть разделена на несколько подтипов, каждый из которых должен содержать общие атрибуты и/или связи. В подтипах могут определяться собственные атрибуты и/или связи характерные для конкретного подтипа.

6.2.4. Получение реляционной схемы из ER-диаграммы

Как мы отмечали выше, большинство средств проектирования ER-диаграмм, среди которых можно выделить такие пакеты, как Power Designer, ErWin, IDEF-Designer и другие, обеспечивают возможность генерации физической модели БД на основе спроектированной концептуальной. Физическая модель может быть преобразована в физическую базу данных практически любого, известного в настоящее время, формата. Процесс преобразования концептуальной модели в физическую, т.е. процесс перехода от ER-схемы к реляционной, можно разделить на этапы.

Этап 1. Каждая простая сущность преобразуется в плоскую таблицу (отношение). Имя сущности становится именем этой таблицы. Простой называется сущность, не являющаяся подтипом и не имеющая подтипов.

Этап 2. Каждый атрибут сущности преобразуется в столбец (атрибут реляционного отношения) с тем же именем. Столбцы реляционной таблицы, соответствующие необязательным атрибутам сущности, могут содержать неопределенные значения, тогда как столбцы, соответствующие обязательным, — не могут.

Этап 3. Атрибуты, входящие в состав уникального идентификатора сущности, преобразуются в первичный ключ таблицы. При наличии альтернативных ключей выбирается наиболее удобный для использования ключ, содержащий меньшее число атрибутов. При этом, если в

состав уникального идентификатора входят связи, в первичный ключ отношения добавляется уникальный идентификатор сущности, находящейся на другом конце связи, с соответствующими именами.

Этап 4. Связи между сущностями типа «один-ко-многим» преобразуются во внешние ключи. Таким образом, создается копия соответствующего уникального идентификатора с конца связи один, и соответствующие столбцы являются внешним ключом. Необязательные связи соответствуют столбцам, допускающим неопределенные значения, обязательные связи — столбцам реляционной таблицы, не допускающим неопределенные значения.

Этап 5. Индексы в таблицах создаются для первичного ключа (уникальный индекс) и внешних ключей. Также допускается возможность создавать индексы на основе других атрибутов сущности.

6.3. CASE-средства

6.3.1. Назначение и классификация CASE-средств

Разработку любой информационной системы следует начинать с проектирования концептуальной модели выбранной предметной области. Значительно сэкономить временные ресурсы и создать нормализованную структуру БД помогают CASE-средства (Computer Aided Software Engineering) автоматизированного проектирования. В более широком понимании CASE-средства предназначены для автоматизации процессов проектирования информационных систем, в этой главе мы остановимся на тех частях CASE-средств, которые отвечают за проектирование структурной части информационных систем, а именно за построение концептуальной и физической модели БД.

Большинство существующих на рынке CASE-средства можно классифицировать по типам и категориям. Так, классификация по типам отражает функциональную ориентацию CASE-средств на те или иные процессы жизненного цикла. Классификация по категориям определяет степень интегрированности по выполняемым функциям и включает отдельные локальные средства, решающие небольшие автономные задачи (tools), набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла ИС (toolkit) и полностью интегрированные средства, поддерживающие весь ЖЦ ИС и связанные общим репозиторием [12].

По типам CASE-средства можно классифицировать следующим образом [12]:

CASE-средства предназначенные для анализа и проектирования, результатом чего являются предварительные спецификации компонентов и интерфейсов информационной системы, алгоритмов и структур данных. К таким системам можно отнести (Vantage Team Builder (Cayenne), Designer/2000 (ORACLE), Silverrun (CSA), CASE.Аналитик (МакроПроджект)).

CASE-средства, назначение которых состоит в построении и анализе концептуальной модели предметной области (Design/IDEF (Meta Software), BPwin (Logic Works));

CASE-средства предназначенные непосредственно для проектирования структуры баз данных на основе спроектированной концептуальной модели предметной области в идеологии конкретной СУБД (ERwin (Logic Works), S-Designor (SDP) DataBase Designer (ORACLE).

CASE-средства реинжиниринга – наиболее современные CASE-средства, предоставляющие возможности проведения широкомасштабного анализа программных кодов и схем баз данных и формирование на их основе различных моделей и предварительных проектных спецификаций (Rational Rose (Rational Software), Object Team (Cayenne)).

6.3.2. Обзор CASE-средств

Рассмотрим наиболее часто встречающиеся на РОСИЙСКОМ рынке CASE-средства, в функции которых входит создание концептуальной модели предметной области и физической структуры БД. При этом будем оперировать сведениями представленными в [12-15].

CASE-средство *Silverrun* – разработка фирмы Computer Systems Advisers, Inc. (США). Silverrun может быть использовано для анализа и проектирования ИС бизнес-класса и ориентировано в большей степени на спиральную модель ЖЦ. Оно применимо для поддержки любой методологии, основанной на раздельном построении функциональной и информационной моделей (диаграмм потоков данных и диаграмм "сущность-связь").

Модуль концептуального моделирования данных (ERX - Entity-Relationship eXpert) обеспечивает построение моделей данных "сущность-связь", не привязанных к конкретной реализации. Этот модуль имеет встроенную экспертную систему, позволяющую создать корректную нормализованную модель данных посредством ответов на содержательные вопросы о взаимосвязи данных. Возможно автоматическое построение модели данных из описаний структур данных. Анализ функциональных зависимостей атрибутов дает возможность про-

верить соответствие модели требованиям третьей нормальной формы и обеспечить их выполнение. Проверенная модель передается в модуль RDM.

Модуль реляционного моделирования (RDM – Relational Data Modeler) позволяет создавать детализированные модели «сущность-связь», предназначенные для реализации в реляционной базе данных. В этом модуле документируются все конструкции, связанные с построением базы данных: индексы, триггеры, хранимые процедуры и т.д. Гибкая изменяемая нотация и расширяемость репозитория позволяют работать по любой методологии. Возможность создавать подсхемы соответствует подходу ANSI SPARC к представлению схемы базы данных. На языке подсхем моделируются как узлы распределенной обработки, так и пользовательские представления. Этот модуль обеспечивает проектирование и полное документирование реляционных баз 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 средства интеграции с наиболее распространенными СУБД: Oracle, Informix, DB2, Ingres, Progress, SQL Server, SQLBase, Sybase. Это позволяет документировать, перепроектировать или переносить на новые платформы уже находящиеся в эксплуатации базы данных и прикладные системы. Таким образом можно полностью определить ядро базы данных с использованием всех возможностей конкретной СУБД: триггеров, хранимых процедур, ограничений ссылочной целостности. Для обмена данными с другими средствами автоматизации проектирования, создания специализированных процедур анализа и проверки проектных спецификаций, составления специализированных отчетов в соответствии с различными стандартами в системе Silverrun имеется различные способы выдачи.

Vantage Team Builder представляет собой интегрированный программный продукт, ориентированный на реализацию каскадной модели жизненного цикла (ЖЦ) программного обеспечения и поддержку полного ЖЦ ПО.

Vantage Team Builder обеспечивает выполнение следующих функций:

– проектирование диаграмм потоков данных, "сущность-связь", структур данных, структурных схем программ и последовательностей экранных форм;

- проектирование диаграмм архитектуры системы - SAD (проектирование состава и связи вычислительных средств, распределения задач системы между вычислительными средствами, моделирование отношений типа "клиент-сервер", анализ использования менеджеров транзакций и особенностей функционирования систем в реальном времени);
- генерация кода программ на языке 4GL целевой СУБД с полным обеспечением программной среды и генерация SQL-кода для создания таблиц БД, индексов, ограничений целостности и хранимых процедур;
- программирование на языке C со встроенным SQL;
- управление версиями и конфигурацией проекта;
- многопользовательский доступ к репозиторию проекта;
- генерация проектной документации по стандартным и индивидуальным шаблонам;
- экспорт и импорт данных проекта в формате CDIF (CASE Data Interchange Format).

При построении модели данных в виде ER-диаграммы в среде Vantage Team Builder выполняется ее нормализация и вводится определение физических имен элементов данных и таблиц, которые будут использоваться в процессе генерации физической схемы данных конкретной СУБД. Обеспечивается возможность определения альтернативных ключей сущностей и полей, составляющих дополнительные точки входа в таблицу (поля индексов), и мощности отношений между сущностями.

Uniface – продукт фирмы Compuware (США) – представляет собой среду разработки крупномасштабных приложений в архитектуре «клиент-сервер» (особенности такой архитектуры будут рассмотрены в главе 11) и имеет в составе компонент Application Model Manager, который поддерживает прикладные ER модели, каждая из которых представляет собой подмножество общей схемы БД с точки зрения данного приложения, и включает соответствующий графический редактор. В список поддерживаемых СУБД входят DB2, VSAM и IMS.

CASE-средство *Designer/2000* фирмы ORACLE является интегрированным CASE-средством, обеспечивающим в совокупности со средствами разработки приложений Developer/2000 поддержку полного ЖЦ ПО для систем, использующих СУБД ORACLE.

Designer/2000 представляет собой семейство методологий и поддерживающих их программных продуктов. Базовая методология Designer/2000 (CASE*Method) - структурная методология проектирования систем, полностью охватывающая все этапы жизненного цикла

информационной системы (ИС) . В соответствии с этой методологией на этапе планирования определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура и план разработки ИС. В процессе анализа строятся модель информационных потребностей (диаграмма «сущность-связь»), диаграмма функциональной иерархии (на основе функциональной декомпозиции ИС), матрица перекрестных ссылок и диаграмма потоков данных.

На этапе проектирования разрабатывается подробная архитектура ИС, проектируется схема реляционной БД и программные модули, устанавливаются перекрестные ссылки между компонентами ИС для анализа их взаимного влияния и контроля за изменениями.

На этапе реализации создается БД, строятся прикладные системы, производится их тестирование, проверка качества и соответствия требованиям пользователей. Создается системная документация, материалы для обучения и руководства пользователей. На этапах эксплуатации и сопровождения анализируются производительность и целостность системы, выполняется поддержка и, при необходимости, модификация ИС;

Designer/2000 обеспечивает графический интерфейс при разработке различных моделей (диаграмм) предметной области. В процессе построения моделей информация о них заносится в репозиторий.

Семейство продуктов *ERWin* от Logic Works предназначено для моделирования и создания баз данных произвольной сложности на основе диаграмм "сущность-связь". В настоящее время ERWin является наиболее популярным пакетом моделирования данных благодаря поддержке широкого спектра СУБД самых различных классов: SQL-серверов (Oracle, Informix, Sybase SQL Server, MS SQL Server, Progress, DB2, SQLBase, Ingress, Rdb и др.) и "настольных" СУБД типа XBase (Clipper, dBASE, FoxPro, MS Access, Paradox и др.).

Информационная модель представляется в виде диаграмм «сущность-связь», отражающих основные объекты предметной области и связи между ними. Дополнительно определяются атрибуты сущностей, характеристики связей. индексы и бизнес-правила, описывающие ограничения и закономерности предметной области. После создания ER-диаграммы пакет автоматически генерирует SQL-код для создания таблиц, индексов и других объектов базы данных. По заданным бизнес-правилам формируются стандартные триггеры БД для поддержки целостности данных, для сложных бизнес-правил можно создавать собственные триггеры, используя библиотеку шаблонов.

Пакет позволяет осуществлять реинжиниринг существующих БД: по SQL-текстам автоматически генерируются ER-диаграммы. Таким

образом пакет поддерживается технология FRE (forward and reverse engineering), последовательность этапов которой приведена ниже:

- импорт с сервера существующей БД;
- автоматическая генерация модели БД;
- модификация модели;
- автоматическая генерация новой схемы и построение физической БД на том же самом или любом другом сервере.

Для разработки клиентской части приложения имеются специальные версии пакета, обеспечивающие интеграцию с такими инструментами как SQLWindows, PowerBuilder, Visual Basic, Delphi. Предлагаются и усеченные версии продукта:

- ERWin/SQL, обеспечивающая лишь прямое проектирование для любых СУБД;
- ERWin/Desktop, поддерживающая технологию FRE только для «настольных» СУБД.

S-Designor представляет собой CASE-средство для проектирования реляционных баз данных. По своим функциональным возможностям и стоимости он близок к CASE-средству ERwin, отличаясь внешне используемой на диаграммах нотацией. S-Designor реализует стандартную методологию моделирования данных и генерирует описание БД для таких СУБД, как ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server и др. Для существующих систем выполняется реинжиниринг БД.

S-Designor совместим с рядом средств разработки приложений (PowerBuilder, Uniface, TeamWindows и др.) и позволяет экспортировать описание БД в репозитории данных средств. Для PowerBuilder выполняется также прямая генерация шаблонов приложений.

Visible Analyst Workbench фирмы Visible Systems представляет собой сетевое многопользовательское средство проектирования информационных систем, базирующееся на репозитарии, хранимом на сервере SQLBase, Oracle или Informix. Пакет основан на методологии Мартина и поддерживает следующие диаграммные техники:

- диаграммы функциональной декомпозиции
- диаграммы потоков данных в нотациях Йодана и Гейна-Сарсона
- диаграммы «сущность-связь»
- структурные карты в нотации Константайна.

Пакет обеспечивает генерацию схем БД для вышеперечисленных СУБД и поддерживает технологию FRE. Имеется возможность экспорта проектов в системы SQLWindows, PowerBuilder и Uniface. К достоинствам пакета может быть отнесено наличие развитых средств верификации проекта, и прежде всего возможностей верти-

кального и горизонтального балансирования диаграмм. Так функциональная и информационная модели сильно коррелированы, что позволяет избавиться от лишних объектов моделей.

Power Designer компании Sybase – средство моделирования масштаба предприятия, объединяющего технологический и бизнес-уровни моделирования для обеспечения максимально эффективного взаимодействия между бизнес и IT-пользователями.

В состав Power Designer входят следующие модули:

- *Process Analyst* - средство для функционального моделирования, поддерживает нотацию Йордона - ДеМарко, Гейна - Сарсона и несколько других. Имеется возможность описать элементы данных (имена, типы, форматы), связанные с потоками данных и хранилищами данных. Эт элементы передаются на следующий этап проектирования, причем хранилища данных могут быть автоматически преобразованыв сущности.
- *Data Analyst (Architect)* - инструмент для построения модели "сущность-связь" и автоматической генерации на ее основе реляционной структуры. Исходные данные для модели "сущность-связь" могут быть получены из DFD-моделей, созданных в модуле Process Analyst. В ER-диаграммах допускаются только бинарные связи, задание атрибутов у связей не поддерживается. Поддерживаются диалекты языка SQL примерно для 30 реляционных СУБД, при этом могут быть сгенерированы таблицы, представления, индексы, триггеры и т.д. В результате порождается SQL-сценарий (последовательность команд CREATE), выполнение которого создает спроектированную схему базы данных. Имеется также возможность установить соединение с СУБД через интерфейс ODBC. Другие возможности: автоматическая проверка правильности модели, расчет размера базы данных, реинжиниринг (построение модельных диаграмм для уже существующих баз данных) и т.д.
- *Application Modeler* - инструмент для автоматической генерации прототипов программ обработки данных на основе реляционных моделей, построенных в Data Analyst. Может быть получен код для Visual Basic, Delphi, а также для таких систем разработки в архитектуре "клиент-сервер" как PowerBuilder, Uniface, Progress и др. Генерация кода осуществляется на основе шаблонов, соответственно управлять генерацией можно за счет изменения соответствующего шаблона.

Power Designer 9.5 и выше позволяет согласовывать объектно-ориентированную и концептуальную модели данных, ориентированную на реляционные СУБД.

Power Designer позволяет генерировать физическую структуру БД на основе спроектированной концептуальной модели для большинства современных СУБД (ORACLE, Informix, Ingres, Sybase, MS SQL Server и др.).

Design/IDEF фирмы Meta Software Corp. (Дания) автоматизирует все этапы проектирования сложных систем различного назначения: формулировку требований и целей проектирования, разработку спецификаций, определение компонентов и взаимодействий между ними, документирование проекта, проверку его полноты и непротиворечивости. Наиболее успешно пакет применяется для описания и анализа деятельности предприятия; он позволяет оценить такую структуру, как единый организм, сочетающий управленческие, производственные и информационные процессы. В основе пакета лежит методология структурного проектирования и анализа сложных систем IDEFO/SADT. Design/IDEF строит иерархические модели сложных систем посредством декомпозиции ее компонентов, поддерживает коллективную разработку IDEF-модели, позволяя в любой момент объединять различные подмодели в единую модель системы, создает словарь данных для хранения всей информации о функциях и структурах данных проекта; формирует 5 типов отчетов, поддерживающих процесс разработки и анализа моделей. Disign/IDEF реализован на платформах MS Windows, Macintosh Plus и выше, Sun Solaris (X Window System), HP9000 модели 700 и 800 (X Window System). Для функционирования Design/CPN требуется: Sun (SPARC), HP9000 модели 700 и 800, X Window System (X11R5), 24 Mb RAM, 32 Mb HDD.

Disign/IDEF также интегрирован с пакетом динамического анализа сложных систем WorkFlow Analyzer и пакетом функционально-стоимостного анализа EasyABC.

6.4. Расчет трудозатрат при проектировании информационных систем и баз данных

6.4.1. Проблемы стандартизации нормативов разработки систем

При проектировании базы данных конкретной предметной области перед разработчиком, помимо технических, возникает ряд других вопросов, напрямую несвязанных с технологией проектирования

базы данных. К этим вопросам можно отнести расчет трудозатрат и определение временных норм на проектирование базы данных и разработку пользовательского приложения, взаимодействующего с этой БД.

На сегодняшний день в литературе отсутствуют какие-либо рекомендации по этой части проектирования БД и систем в целом, единственным стандартом на основе которого проводятся расчеты по трудоемкости разработки систем, является нормативы из действующего на настоящий момент отраслевого «Сборника временных норм на работы по ведению Государственного мониторинга геологической среды, информационной деятельности, цифровому картографированию» (Утверждены Министерством природных ресурсов РФ 27.12.2000г.) [/16/17]. Согласно рекомендациям, предложенным в этом сборнике, расчет трудозатрат производится следующим образом.

6.4.2. Механизм определения трудозатрат

Объем структуры базы данных информационной системы (ИС) согласно [/16/17] определяется:

- количеством объектов системы;
- количеством атрибутов, входящих в 1 объект;
- степенью связанности объектов и атрибутов ИС.

Под сущностью в данном случае будем понимать объект предметной области (одна и более электронных таблиц), элемент информационной системы, примерами сущностей являются законодательная инициатива, входящий документ, исходящий документ, организация, сотрудники организаций и др.

Под атрибутом сущности (полем электронной таблицы) здесь будем понимать простейший элемент информационной системы, формирующийся вручную или посредством справочников (классификаторов) из элементов первичной однородной фактографической информации, разделенной по назначению.

Жизненный цикл разработки информационной системы в целом включает:

- анализ предметной области
- проектирование ИС в идеологии конкретной СУБД и создание пользовательского интерфейса
- создание запросов и отчётов для вывода информации в нужном виде
- расширение функциональных возможностей ИС
- тестирование ИС

- внедрение ИС в эксплуатацию

Процедура создания БД описывается выражением определяющем общее количество полей в наименьшей величине–измерителе.

Количество полей = $2N*10K_1*5K_2$, (1),

где N – коэффициент, отражающий количество объектов ИС;

K_1 – коэффициент, отражающий количество атрибутов на 1 объект;

K_2 – коэффициент, отражающий количество связей с другими объектами.

5) Нормализованной величиной при разработке ИС является величина, характеризуемая количеством формируемых полей, входящих в создаваемые таблицы посредством установленных связей При значениях коэффициентов, равных единице, величина, выражающая их количество равна 100.

Трудоемкость работ рассчитывается с учетом категорий сложности создания информационных систем. В соответствии с нормативами трудозатраты основных исполнителей по созданию ИС численно равны нормам длительности выполнения этой работы (таблица 1)

Таблица 1. Нормы длительности создания ИС с применением СУБД.(Измеритель –100 полей)

Категория сложности	**Значение нормы, смен**	**Характеристика категории**
1 Простая	0,136	Создание информационных систем с использованием: – типовых программных средств с применением «мастера (БД)»; – количество прикладных программ индивидуально, но не более трех; – количество полей электронных таблиц до 90000; – разработанных БД в других организациях и заимствованных без изменения; – наработок созданных ранее собственных элементов БД.
2 Средней сложности	0,194	Создание информационных систем с использованием: – типовых программных средств без применения «мастера БД»; – разработанных БД в других организациях и заимствованных с изменениями, определяемых организацией разработчиком, но без изменения

Категория сложности	Значение нормы, смен	Характеристика категории
		структуры БД; – количество прикладных программ от трех до десяти; – количество полей электронных таблиц от 90000 до 200000; – заимствованных элементов созданных БД
3 Сложная	0,369	Создание информационных систем с использованием: – языка программирования, совместимого с данной СУБД; – количество прикладных программ не ограничено; – количество полей электронных таблиц от 200000 до 500000; – наработок и создаваемых элементов, не имеющих аналогов и разрабатываемых впервые.

Рассмотрим расчет трудоемкости на примере. Пусть даны исходные данные для расчета трудоемкости разработки программного комплекса, описывающего документооборот организации:

- в базе данных следует определить не менее 20 составных сущностей;
- среднее число атрибутов (таких как входящий № документа, входящая дата, тип документа, краткое содержание, резолюция и т.д.) брать не менее 35 на одну составную сущность.

Расчет количества полей производится согласно формуле (1):

Количество полей = 2*20*10*35*5*4= 280000

Таким образом, исходя из коэффициентов, представленных в таблице 1 создаваемая система определенно принадлежит к классу сложных систем. Тогда расчет трудоемкости с учетом нормативов таблицы 1 будет проведен следующим образом:

Количество человеко-смен на создание Информационной системы = 280000*0,369/100=1033,2 ч/с.

Исходя из полученных данных, мы имеем возможность определить количество исполнителей при известных временных ограничениях, а также размер заработной платы исполнителей. Так для нашего примера, при необходимости разработать систему за 12 месяцев при 40-часовой рабочей неделе получим примерно 250 рабочих смен. Таким образом, для разработки системы потребуется 1033,2/258 = 4 исполнителя.

Следует отметить что расчет трудоемкости в данном случае проводится в целом на создание информационной системы, а не только структурной части базы данных.

Контрольные вопросы

1. Дайте определения основных понятий реляционной модели данных.
2. Приведите требования удовлетворения отношений нормальным формам (1НФ, 2НФ, 3НФ).
3. Опишите технологию семантического моделирования предметной области в терминах ER-модели.
4. Дайте определение понятий сущности и связи в ER-модели.
5. Перечислите и кратко охарактеризуйте наиболее известные CASE-средства.
6. Опишите основные функции приведенных в обзоре CASE- средств.
7. Поясните механизм определения трудозатрат на создание информационной системы.

7. РЕЛЯЦИОННЫЕ ЯЗЫКИ МАНИПУЛИРОВАНИЯ ДАННЫМИ SQL И QBE

7.1. Язык SQL

7.1.1. История развития языка

Как отмечалось выше, при работе с БД необходимо осуществлять доступ к данным и управлять ими. Таким образом, важным требованием к реляционным СУБД является наличие языка, позволяющего выполнять все необходимые пользователям операции. В этом разделе остановимся на более подробном описании языков манипулирования данными.

В процессе эволюции СУБД наибольшее распространение получили два языка управления и манипулирования данными:

– язык структурированных запросов **SQL** (Structured Query Language), созданный в исследовательской лаборатории фирмы IBM в Сан-Хосе в середине 70-х годов XX века. В первоначальном варианте язык получил название **SEQUEL** (Structured English Query Language). Стандарт SQL был впервые опубликован в 1986 г. и обновлялся в 1989, 1992 и 1999 гг.;

– запрос по образцу **QBE** (Query-by-Example), созданный фирмой IBM в Йорктаун-Хейтсе.

Наибольшее распространение в процессе эволюции СУБД получил язык SQL, хотя на начальном этапе более перспективным считался язык QBE, как наиболее близкий к пользовательскому интерфейсу. В настоящее время язык SQL, расширенный возможностью визуального проектирования запросов с использованием технологии QBE, используется в большинстве современных СУБД.

Помимо операторов формулирования запросов к базе данных и возможностей манипулирования данными, язык содержит:

– средства определения схемы БД и манипулирования схемой;

– операторы для определения ограничений целостности и триггеров;

– средства определения представлений БД;

– средства авторизации доступа к отношениям и их полям;

– средства управления транзакциями.

Различные СУБД предоставляют возможность выполнения программных конструкций (объектов), записанных в разных диалектах и процедурных расширениях языка SQL: «Watcom-Sql», «Transact-Sql», «PL-Sql» и т.д. Существует достаточное количество публикаций по

различным диалектам языка SQL, рассмотрим инструкции языка SQL в приближении к общепринятому базовому стандарту SQL-92, а также ряд отличий, специфичных для синтаксиса, используемого в среде MS Access (стандарт Microsoft Jet SQL). Язык SQL ядра базы данных Microsoft Jet в основном соответствует стандарту ANSI-89 (уровень 1). Однако некоторые средства ANSI SQL не реализованы в языке SQL ядра Microsoft Jet. И наоборот, язык SQL ядра Microsoft Jet использует зарезервированные слова и средства, не поддерживаемые ANSI SQL.

7.1.2. Синтаксис команд SQL

В большинстве работ по описанию команд SQL авторы придерживаются общего синтаксиса написания SQL-предложений:

– в описании команд слова, написанные прописными латинскими буквами, являются зарезервированными словами SQL;

– фрагменты SQL-предложений, заключенные в фигурные скобки и разделенные символом «|», являются альтернативными. При формировании соответствующей команды для конкретного случая необходимо выбрать одну из них;

– фрагмент описываемого SQL-предложения, заключенный в квадратные скобки **[ ],** имеет необязательный характер и может не использоваться;

– многоточие …, стоящее перед закрывающейся скобкой, говорит о том, что фрагмент, указанный в этих скобках, может быть повторен;

7.1.3. Описание команд SQL

Выборка записей

Инструкция **SELECT**. При выполнении инструкции SELECT СУБД находит указанную таблицу или таблицы, извлекает заданные столбцы, выделяет строки, соответствующие условию отбора, и сортирует или группирует результирующие строки в указанном порядке в виде набора записей.

Синтаксис команды:

SELECT [предикат] { * | таблица.* | [таблица.]поле_1
[AS псевдоним_2] [, [таблица.]поле_2[AS псевдоним_2] [, ...]]}
FROM выражение [, ...]
[WHERE...]
[GROUP BY...]
[HAVING...]
[ORDER BY...]

где предикат — один из следующих предикатов отбора: ALL, DISTINCT, DISTINCTROW, TOP. Данные ключевые слова используются для ограничения числа возвращаемых записей. Если они отсутствуют, по умолчанию используется предикат ALL;

* указывает, что результирующий набор записей будет содержать все поля заданной таблицы или таблиц. Следующая инструкция отбирает все поля из таблицы «Студенты»: SELECT * FROM Студенты;

таблица — имя таблицы, из которой должны быть выбраны записи;

поле_1, поле_2 – имена полей, из которых должны быть отобраны данные;

псевдоним_1, псевдоним_2 — ассоциации, которые станут заголовками столбцов вместо исходных названий полей в таблице;

выражение — имена одной или нескольких таблиц, которые содержат необходимые для отбора записи;

предложение GROUP BY в SQL-предложении объединяет записи с одинаковыми значениями в указанном списке полей в одну запись. Если инструкция SELECT содержит статистическую функцию SQL, например, Sum или Count, то для каждой записи будет вычислено итоговое значение;

предложение HAVING определяет, какие сгруппированные записи, выданные в результате выполнения запроса, отображаются при использовании инструкции SELECT с предложением GROUP BY. После того как записи результирующего набора будут сгруппированы с помощью предложения GROUP BY, предложение HAVING отберет те из них, которые удовлетворяют условиям отбора, указанным в предложении HAVING;

предложение ORDER BY позволяет отсортировать записи, полученные в результате запроса, в порядке возрастания или убывания на основе значений указанного поля или полей.

Следует отметить, что инструкции SELECT не изменяют данные в базе данных.

Ниже приведен минимальный синтаксис инструкции SELECT:

SELECT поля FROM таблица

Если несколько таблиц, включенных в предложение FROM, содержат одноименные поля, перед именем такого поля следует ввести имя таблицы и оператор « . » (точка). Предположим, что поле «Номер_группы» содержится в таблицах «Студенты» и «Группы». Следующая инструкция SQL отберет поле «Номер_группы» и «ФИО_студента» из таблицы «Студенты» и «ФИО_куратора» из таблицы «Группы», при номере группы равном 432-1:

SELECT Группы.Номер_группы, Группы.ФИО_куратора, Студенты.ФИО_студента

FROM Группы, Студенты

WHERE Группы.Номер_группы = Студенты.Номер_группы AND

На рис. 7.1 приведен пример выполнения данного запроса

Таблицы БД

СТУДЕНТЫ

Номер_зачетной книжки	ФИО_студента	Дата рождения	Место рождения	Номер_группы
1992412-11	Карасев А.А.	27.08.75	г. Чита	412-1
1992432-11	Данилов О. В.	27.08.75	г. Алма-Ата	432-1
1992432-12	Раевский А. И.	20.05.75	г. Бишкек	432-1
1992432-22	Глазов О.А	04.07.75	г. Киров	432-1

ГРУППЫ

Номер_Группы	ФИО_куратора
412-1	Самойлов С.С.
432-1	Авдеев Р.М

Результат выполнения запроса

Номер_группы	ФИО_куратора	ФИО_студента
432-1	Авдеев Р.М	Данилов О. В.
432-1	Авдеев Р.М	Раевский А. И.
432-1	Авдеев Р.М	Глазов О.А

Рис. 7.1 Пример выполнения запроса на выборку

Помимо обычных знаков сравнения (=,<,>,<=,>=, <>) в языке SQL в условии отбора используются ряд ключевых слов:

Is not null – выбрать только непустые значения;

Is null – выбрать только пустые значения;

Between … And – определяет принадлежность значения выражения указанному диапазону.

Синтаксис

выражение [Not] Between значение_1 And значение_2 ,

где выражение – выражение, определяющее поле, значение которого проверяется на принадлежность к диапазону.

значение_1, значение_2 – выражения, задающие границы диапазона.

Если значение поля, определенного в аргументе выражение, попадает в диапазон, задаваемый аргументами значение_1 и значение_2

(включительно), оператор Between...And возвращает значение True; в противном случае возвращается значение False. Логический оператор Not позволяет проверить противоположное условие (что выражение находится за пределами диапазона, заданного с помощью аргументов значение_1 и значение_2).

Оператор Between...And часто используют для проверки, попадает ли значение поля в указанный диапазон чисел. В следующем примере выдается список студентов получающих стипендию от 800 до 900 рублей:

```
SELECT ФИО_студента, Размер_стипендии
FROM Студенты
WHERE Размер_стипендии Between 800 And 900
```

На рис. 7.2 приведен результат выполнения запроса.

СТУДЕНТЫ

Номер_зачет-ной книжки	ФИО_студента	Размер_стипендии
1992412-11	Карасев А.А.	900
1992432-11	Данилов О. В.	800
1992432-12	Раевский А. И.	950
1992432-22	Глазов О.А	850

Результирующий набор данных

ФИО_студента	Размер_стипендии
Карасев А.А.	900
Данилов О. В.	800
Глазов О.А	850

Рис. 7.2 Результат выполнения запроса на выборку с использованием операторов Between...And

Если выражение, значение_1 или значение_2 имеет значение Null, оператор Between...And возвращает значение Null.

Оператор Like – используется для сравнения строкового выражения с образцом.

Синтаксис

выражение Like «образец»,

где

выражение – выражение SQL, используемое в предложении WHERE.

образец – строка, с которой сравнивается выражение.

Оператор Like используется для нахождения в поле значений, соответствующих указанному образцу. Для аргумента образец можно задавать полное значение (например, Like «Иванов») или использовать

подстановочные знаки для поиска диапазона значений (например, Like "Ив*").

В таблице 7.1 приведен перечень подстановочных символов и пример их использования в языке Jet SQL, согласно документации Microsoft.

Таблица 7.1. Параметры оператора Like

Тип совпадения	**Образец**	**Совпадение (True)**	**Несовпадение (False)**
Несколько символов	a*a	aa, aBa, aBBBa	aBC
ab	abc, AABB, Xab	aZb, bac	
Специальный символ	a[*]a	a*a	aaa
Несколько символов	ab*	abcdefg, abc	cab, aab
Одиночный символ	a?a	aaa, a3a, aBa	aBBBa
Одиночная цифра	a#a	a0a, a1a, a2a	aaa, a10a
Диапазон символов	[a-z]	f, p, j	2, &
Вне диапазона	[!a-z]	9, &, %	b, a
Не цифра	[!0-9]	A, a, &, ~	0, 1, 9
Комбинированное выражение	a[!b-m]#	An9, az0, a99	abc, aj0

Внутреннее соединение

Операция **INNER JOIN** объединяет записи из двух таблиц, если связующие поля этих таблиц содержат одинаковые значения. Синтаксис операции:

FROM таблица_1 INNER JOIN таблица_2 ON таблица_1.поле_1 оператор таблица_2.поле_2

где таблица_1, таблица_2 — имена таблиц, записи которых подлежат объединению;

поле_1, поле_2 — имена объединяемых полей. Поля должны иметь одинаковый тип данных и содержать данные одного рода, однако эти поля могут иметь разные имена;

оператор — любой оператор сравнения: "=," "<," ">," "<=," ">=," или "<>".

Операцию INNER JOIN можно использовать в любом предложении FROM. Это самые обычные типы связывания. Они объединяют записи двух таблиц, если связующие поля обеих таблиц содержат одинаковые значения. Предыдущий пример использования команды

SELECT можно записать с использованием конструкции INNER JOIN следующим образом:

```
SELECT Группы.Номер_группы, Группы.ФИО_куратора, Студенты.ФИО_студента
FROM Группы INNER JOIN Студенты
ON Группы.Номер_группы = Студенты.Номер_группы;
```

Внешнее соединение

Операции **LEFT JOIN, RIGHT JOIN** объединяют записи исходных таблиц при использовании в любом предложении FROM.

Операция LEFT JOIN используется для создания внешнего соединения, при котором все записи из первой (левой) таблицы включаются в результирующий набор, даже если во второй (правой) таблице нет соответствующих им записей.

Операция RIGHT JOIN используется для создания внешнего объединения, при котором все записи из второй (правой) таблицы включаются в результирующий набор, даже если в первой (левой) таблице нет соответствующих им записей.

Синтаксис операции:

FROM таблица_1 [**LEFT | RIGHT**] **JOIN** таблица_2
ON таблица_1.поле_1 оператор таблица_2.поле_2

Например, операцию LEFT JOIN можно использовать с таблицами «Студенты» (левая) и «Задолженность_за_обучение» (правая) для отбора всех студентов, в том числе тех, которые не являются задолжниками:

```
SELECT Студенты.ФИО_студента,
Задолженность_за_обучение.Сумма_задолженности
FROM Студенты LEFT JOIN Задолженность_за_обучение
ON Студенты.Номер_зачетной_книжки =
 Задолженность_за_обучение.Номер_зачетной_книжки;
```

В этом примере поле «Номер_зачетной_книжки» используется для объединения таблиц, однако, не включается в результат выполнения запроса, поскольку не включено в инструкцию SELECT. Чтобы включить связующее поле (в данном случае поле «Номер_зачетной_книжки») в результат выполнения запроса, его имя необходимо включить в инструкцию SELECT.

Важно отметить, что операции LEFT JOIN или RIGHT JOIN могут быть вложены в операцию INNER JOIN, но операция INNER JOIN не может быть вложена в операцию LEFT JOIN или RIGHT JOIN.

Перекрестные запросы

В некоторых СУБД (в частности в MS Access) существует такой вид запросов как перекрестный. В перекрестном запросе отображаются результаты статистических функций – суммы, средние значения и др., а также количество записей. При этом подсчет выполняется по данным из одного полей таблицы. Результаты группируются по двум наборам данных, один из которых расположен в левом столбце таблицы, а другой в заголовке таблицы. Например, при необходимости вычислить средний бал студентов за семестр, обучающихся на разных кафедрах, необходимо реализовать перекрестный запрос в результате выполнения которого будет создана таблица, в которой заголовками строк будут служить номер семестра, заголовками столбцов – названия кафедр, а в полях таблицы будут рассчитан средний бал.

Для создания перекрестного запроса необходимо использовать следующую инструкцию:

TRANSFORM статистическая_функция
инструкция_SELECT
PIVOT поле [IN (значение_1[, значение_2[, ...]])], где

статистическая_функция – Статистическая функция SQL, обрабатывающая указанные данные.

Инструкция_SELECT – запрос на выборку

поле – Поле или выражение, которое содержит заголовки столбцов для результирующего набора.

значение_1, значение_2 Фиксированные значения, используемые при создании заголовков столбцов.

Составим SQL-запрос реализующий описанный выше пример. В качестве исходного набора данных используется таблица Успеваемость (рис. 7.3).

Успеваемость : таблица

НОМЕР_ЗАЧЕТНОЙ_КНИЖКИ	Семестр	Оценка	Кафедра
102	1	3	АСУ
102	2	3	АСУ
102	3	4	АСУ
102	4	4	АСУ
110	1	5	АОИ
110	2	3	АОИ
110	3	3	АОИ
110	4	4	АОИ
110	5	4	АОИ

Рис. 7.3 Таблица УСПЕВАЕМОСТЬ

В результате выполнения следующего перекрестного SQL-запроса:

```
TRANSFORM AVG(Успеваемость.Оценка) AS Сред_бал
SELECT Успеваемость.Семестр
FROM Успеваемость
GROUP BY Успеваемость.Семестр
PIVOT Успеваемость.Кафедра
```

формируется следующая таблица (рис. 7.4)

SQL_запрос : перекрестный запрос

	Семестр	АОИ	АСУ
▶	1	5	3
	2	3	3
	3	3	4
	4	4	4
	5	4	

Рис. 7.4 Результат выполнения перекрестного запроса

Таким образом, когда данные сгруппированы с помощью перекрестного запроса, можно выбирать значения из заданных столбцов или выражений как заголовки столбцов. Это позволяет просматривать данные в более компактной форме, чем при работе с запросом на выборку.

Подчиненные запросы

Часто возникает ситуация, когда желаемый результат нельзя получить с помощью одного SQL-запроса. Одним из способов решения такой задачи является использование подчиненных запросов в составе главного SQL-запроса. Подчиненным SQL-запросом называют инструкцию SELECT, включаемую в инструкции SELECT, SELECT...INTO, INSERT...INTO, DELETE или UPDATE или в другой подчиненный запрос. Подчиненный запрос может быть создан одним из трех способов:

Синтаксис

```
сравнение [ANY | ALL | SOME] (инструкцияSQL)
выражение [NOT] IN (инструкцияSQL)
[NOT] EXISTS (инструкцияSQL), где
```

сравнение – выражение и оператор сравнения, который сравнивает выражение с результатами подчиненного запроса.

выражение – выражение, для которого проводится поиск в результирующем наборе записей подчиненного запроса.

инструкцияSQL – инструкция SELECT, заключенная круглые скобки.

Подчиненный запрос можно использовать вместо выражения в списке полей инструкции SELECT или в предложениях WHERE и HAVING. Инструкция SELECT используется в подчиненном запросе для задания набора конкретных значений, вычисляемых в выражениях предложений WHERE или HAVING.

Предикаты ANY или SOME, являющиеся синонимами, используются для отбора записей в главном запросе, которые удовлетворяют сравнению с записями, отобранными в подчиненном запросе. В следующем примере отбираются все студенты, средний бал которых за семестр больше 4.

```
SELECT * FROM Студенты
WHERE Номер_зачетной_книжки = ANY
(SELECT Номер_зачетной_книжки FROM Успеваемость
WHERE оценка > 4)
```

Предикат ALL используется для отбора в главном запросе только тех записей, которые удовлетворяют сравнению со всеми записями, отобранными в подчиненном запросе. Если в предыдущем примере предикат ANY заменить предикатом ALL, результат запроса будет включать только тех студентов чей средний бал больше 4. Это условие является значительно более жестким.

Предикат IN используется для отбора в главном запросе только тех записей, которые содержат значения, совпадающие с одним из отобранных подчиненным запросом. Следующий пример возвращает сведения о всех студентах, средний бал которых за семестр больше 4.

```
SELECT * FROM Студенты
WHERE Номер_зачетной_книжки in
(SELECT Номер_зачетной_книжки FROM Успеваемость
WHERE оценка > 4)
```

Предикат NOT IN используется для отбора в главном запросе только тех записей, которые содержат значения, не совпадающие ни с одним из отобранных подчиненным запросом.

Предикат EXISTS (с необязательным зарезервированным словом NOT) используется в логическом выражении для определения того, должен ли подчиненный запрос возвращать какие-либо записи.

В подчиненном запросе можно использовать псевдонимы таблиц для ссылки на таблицы, перечисленные в предложении FROM, расположенном вне подчиненного запроса. В следующем примере отбираются фамилии и имена студентов, чья стипендия равна или больше средней стипендии студентов, обучающихся в той же группе. В данном примере таблица СТУДЕНТЫ получает псевдоним С1:

```
SELECT Фамилия,
Имя, Номер_группы, Стипендия
FROM СТУДЕНТЫ AS С1
WHERE Стипендия >=
(SELECT Avg(Стипендия)
FROM СТУДЕНТЫ
WHERE С1.Номер_группы = СТУДЕНТЫ.Номер_группы) Order by Номер_группы;
```

В последнем примере зарезервированное слово AS не является обязательным.

Некоторые подчиненные запросы можно использовать в перекрестных запросах как предикаты (в предложении WHERE). Подчиненные запросы, используемые для вывода результатов (в списке SELECT), нельзя использовать в перекрестных запросах.

Создание новой таблицы

Инструкция **CREATE TABLE** создает новую таблицу и используется для описания ее полей и индексов. Если для поля добавлено ограничение NOT NULL, то при добавлении новых записей это поле должно содержать допустимые данные. Синтаксис

CREATE TABLE таблица (поле_1 тип [(размер)]
[NOT NULL] [индекс_1] [, поле_2 тип [(размер)]
[NOT NULL] [индекс_2] [, ...]] [, CONSTRAINT составной Индекс [, ...]]),

где таблица — имя создаваемой таблицы;

поле_1, поле_2 — имена одного или нескольких полей, создаваемых в новой таблице. Таблица должна содержать хотя бы одно поле;

тип — тип данных поля в новой таблице;

размер — размер поля в символах (только для текстовых и двоичных полей);

индекс_1, индекс_2 — предложение CONSTRAINT, предназначенное для создания простого индекса;

составной Индекс — предложение CONSTRAINT, предназначенное для создания составного индекса.

В следующем примере создается новая таблица с двумя полями:

CREATE TABLE Студенты (Номер_зачетной_книжки integer PRYMARY KEY, ФИО_студента TEXT (50), Место_рождения TEXT (50));

В результате выполнения этого запроса будет создана таблица со следующей схемой (рис. 7.5):

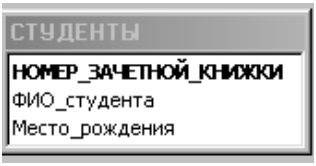

Рис. 7.5 Схема таблицы СТУДЕНТЫ

Предложение CONSTRAINT

Предложение **CONSTRAINT** используется в инструкциях ALTER TABLE и CREATE TABLE для создания или удаления индексов. Существуют два типа предложений CONSTRAINT: для создания простого индекса (по одному полю) и для создания составного индекса (по нескольким полям). Синтаксис:

Простой индекс:

CONSTRAINT имя {PRIMARY KEY|UNIQUE | NOT NULL]}

Составной индекс:

CONSTRAINT имя

{PRIMARY KEY (ключевое_1[, ключевое_2 [, ...]]) |

UNIQUE (уникальное_1[, уникальное_2 [, ...]]) |

NOT NULL (непустое_1[, непустое_2 [, ...]]) |

FOREIGN KEY (ссылка_1[, ссылка_2 [, ...]])

REFERENCES внешняя Таблица [(внешнее Поле_1 [, внешнее Поле_2 [, ...]])]},

где имя — имя индекса, который следует создать;

ключевое_1, ключевое_2 — имена одного или нескольких полей, которые следует назначить ключевыми;

уникальное_1, уникальное_2 — имена одного или нескольких полей, которые следует включить в уникальный индекс;

непустое_1, непустое_2 — имена одного или нескольких полей, в которых запрещаются значения Null;

ссылка_1, ссылка_2 — имена одного или нескольких полей, включенных во внешний ключ, которые содержат ссылки на поля в другой таблице;

внешняя Таблица — имя внешней таблицы, которая содержит поля, указанные с помощью аргумента внешнееПоле;

внешнее Поле_1, внешнее Поле_2 — имена одного или нескольких полей во внешней Таблице, на которые ссылаются поля, указанные с помощью аргумента ссылка_1, ссылка_2. Это предложение можно опустить, если данное поле является ключом внешней Таблицы.

Предложение CONSTRAINT позволяет создать для поля индекс одного из двух описанных ниже типов:

1) для создания уникального индекса используется зарезервированное слово UNIQUE. Это означает, что в таблице не может быть двух записей, имеющих одно и то же значение в этом поле. Уникальный индекс создается для любого поля или любой группы полей. Если в таблице определен составной уникальный индекс, то комбинация значений включенных в него полей должна быть уникальной для каждой записи таблицы, хотя отдельные поля и могут иметь совпадающие значения;

2) для создания ключа таблицы, состоящего из одного или нескольких полей, используются зарезервированные слова PRIMARY KEY. Все значения ключа таблицы должны быть уникальными и не значениями Null. Кроме того, в таблице может быть только один ключ.

Для создания внешнего ключа можно использовать зарезервированную конструкцию **FOREIGN KEY**. Если ключ внешней таблицы состоит из нескольких полей, необходимо использовать предложение CONSTRAINT. При этом следует перечислить все поля, содержащие ссылки на поля во внешней таблице, а также указать имя внешней таблицы и имена полей внешней таблицы, на которые ссылаются поля, перечисленные выше, причем в том же порядке. Однако, если последние поля являются ключом внешней таблицы, то указывать их необязательно, поскольку ядро базы данных считает, что в качестве этих полей следует использовать поля, составляющие ключ внешней таблицы.

В следующем примере создается таблица ЗАДОЛЖЕННОСТЬ_ЗА_ОБУЧЕНИЕ с единственным полем НОМЕР_ЗАЧЕТ-

НОЙ_КНИЖКИ и внешним ключом f1_i, связанным с полем НОМЕР_ЗАЧЕТНОЙ_КНИЖКИ в таблице СТУДЕНТЫ:

CREATE TABLE Задолженность_за_обучение

(Код_задолженности integer PRIMARY KEY, Номер_зачетной_книжки integer, CONSTRAINT f1_i FOREIGN KEY (Номер_зачетной_книжки) REFERENCES Студенты (Номер_зачетной_книжки));

Внешний вид схемы БД, состоящей из таблиц СТУДЕНТЫ и ЗАДОЛЖЕННОСТЬ_ЗА_ОБУЧЕНИЕ представлен на рис. 7.6

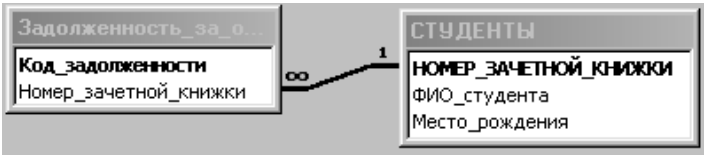

Рис. 7.6 Схема данных

Изменение структуры таблицы

Инструкция ALTER TABLE изменяет структуру таблицы, созданной с помощью инструкции CREATE TABLE. Синтаксис:

ALTER TABLE таблица {ADD {COLUMN поле тип[(размер)] [NOT NULL]

[CONSTRAINT индекс] | CONSTRAINT составной Индекс} |

DROP {COLUMN поле I CONSTRAINT имя Индекса} }

где таблица — имя изменяемой таблицы;

поле — имя поля, добавляемого в таблицу или удаляемого из нее;

тип — тип данных поля;

размер — размер поля;

индекс — индекс для поля;

составной Индекс — описание составного индекса, добавляемого к таблице;

имя Индекса — имя составного индекса, который следует удалить.

С помощью инструкции ALTER TABLE существующую таблицу можно изменить несколькими способами:

1) добавить новое поле в таблицу с помощью предложения ADD COLUMN. В этом случае необходимо указать имя поля, его тип и размер. Например, следующая инструкция добавляет в таблицу «Студенты» текстовое поле «Примечания» длиной 50 символов:

ALTER TABLE Студенты ADD COLUMN Примечания TEXT(50)

Если для поля добавлено ограничение NOT NULL, то при добавлении новых записей это поле должно содержать допустимые данные;

2) добавить составной индекс с помощью зарезервированных слов ADD CONSTRAINT;

3) удалить поле с помощью зарезервированных слов DROP COLUMN. В этом случае необходимо указать только имя поля;

4) удалить составной индекс с помощью зарезервированных слов DROP CONSTRAINT. В этом случае указывается только имя составного индекса, следующее за зарезервированным словом CONSTRAINT.

Создание индекса с помощью инструкции CREATE INDEX

CREATE INDEX создает новый индекс для существующей таблицы. Синтаксис команды:

CREATE [UNIQUE] **INDEX** индекс
ON таблица (поле [ASC|DESC][, поле [ASC|DESC], ...])
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]

где индекс — имя создаваемого индекса;

таблица — имя существующей таблицы, для которой создается индекс;

поле — имена одного или нескольких полей, включаемых в индекс. Для создания простого индекса, состоящего из одного поля, вводится имя поля в круглых скобках сразу после имени таблицы. Для создания составного индекса, состоящего из нескольких полей, перечисляются имена всех этих полей. Для расположения элементов индекса в убывающем порядке используется зарезервированное слово DESC; в противном случае будет принят порядок по возрастанию.

Чтобы запретить совпадение значений индексированных полей в разных записях, используется зарезервированное слово UNIQUE. Необязательное предложение WITH позволяет задать условия на значения. Например:

– с помощью параметра DISALLOW NULL можно запретить значения Null в индексированных полях новых записей;

– параметр IGNORE NULL позволяет запретить включение в индекс записей, имеющих значения Null в индексированных полях;

– зарезервированное слово PRIMARY позволяет назначить индексированные поля ключом. Такой индекс по умолчанию является уникальным, следовательно, зарезервированное слово UNIQUE можно опустить.

Удаление таблицы/индекса

Инструкция **DROP** удаляет существующую таблицу из базы данных или удаляет существующий индекс из таблицы. Синтаксис:

DROP {TABLE таблица | INDEX индекс ON таблица}

где таблица — имя таблицы, которую следует удалить или из которой следует удалить индекс;

индекс — имя индекса, удаляемого из таблицы.

Прежде чем удалить таблицу или удалить из нее индекс, необходимо ее закрыть. Следует отметить, что таблица удаляется из базы данных безвозвратно.

Удаление записей

Инструкция **DELETE** создает запрос на удаление записей из одной или нескольких таблиц, перечисленных в предложении FROM, которые удовлетворяют предложению WHERE. Синтаксис команды:

DELETE [Таблица.*]
FROM таблица
WHERE условие Отбора

где Таблица — необязательное имя таблицы, из которой удаляются записи;

таблица — имя таблицы, из которой удаляются записи;

условие Отбора — выражение, определяющее удаляемые записи.

С помощью инструкция DELETE можно осуществлять удаление большого количества записей. Данные из таблицы также можно удалить и с помощью инструкции DROP, однако при таком удалении теряется структура таблицы. Если же применить инструкцию DELETE, удаляются только данные. При этом сохраняются структура таблицы и все остальные ее свойства, такие, как атрибуты полей и индексы.

Запрос на удаление удаляет записи целиком, а не только содержимое указанных полей. Нельзя восстановить записи, удаленные с помощью запроса на удаление. Чтобы узнать, какие записи будут удалены, необходимо посмотреть результаты запроса на выборку, использующего те же самые условие отбора в предложении Where, а затем выполнить запрос на удаление.

Добавление записей

Инструкция **INSERT INTO** добавляет запись или записи в таблицу. Синтаксис команды:

а) запрос на добавление нескольких записей:
INSERT INTO назначение [(поле_1[, поле_2[, ...]])]
SELECT [источник.]поле_1[, поле_2[, ...]
FROM выражение

б) запрос на добавление одной записи:
INSERT INTO назначение [(поле_1[, поле_2[, ...]])]
VALUES (значение_1[, значение_2[, ...])

где назначение — имя таблицы или запроса, в который добавляются записи;

источник — имя таблицы или запроса, откуда копируются записи;

поле_1, поле_2 — имена полей для добавления данных, если они следуют за аргументом «Назначение»; имена полей, из которых берутся данные, если они следуют за аргументом источник;

выражение — имена таблицы или таблиц, откуда вставляются данные. Это выражение может быть именем отдельной таблицы или результатом операции INNER JOIN, LEFT JOIN или RIGHT JOIN, а также сохраненным запросом;

значение_1, значение_2 — значения, добавляемые в указанные поля новой записи. Каждое значение будет вставлено в поле, занимающее то же положение в списке: значение_1 вставляется в поле_1 в новой записи, значение_2 — в поле_2 и т.д. Каждое значение текстового поля следует заключать в кавычки (' '), для разделения значений используются запятые.

Инструкцию INSERT INTO можно использовать для добавления одной записи в таблицу с помощью запроса на добавление одной записи, описанного выше. В этом случае инструкция должна содержать имя и значение каждого поля записи. Нужно определить все поля записи, в которые будет помещено значение, и значения для этих полей. Если поля не определены, в недостающие столбцы будет вставлено значение по умолчанию или значение Null. Записи добавляются в конец таблицы.

Инструкцию INSERT INTO можно также использовать для добавления набора записей из другой таблицы или запроса с помощью предложения SELECT ... FROM, как показано выше в запросе на добавление нескольких записей. В этом случае предложение SELECT определяет поля, добавляемые в указанную таблицу «Назначение».

Инструкция INSERT INTO является необязательной, однако, если она присутствует, то должна находиться перед инструкцией SELECT.

Запрос на добавление записей копирует записи из одной или нескольких таблиц в другую таблицу. Таблицы, которые содержат добавляемые записи, не изменяются.

Вместо добавления существующих записей из другой таблицы, можно указать значения полей одной новой записи с помощью предложения VALUES. Если список полей опущен, предложение VALUES должно содержать значение для каждого поля таблицы; в противном случае инструкция INSERT не будет выполнена. Можно использовать дополнительную инструкцию INSERT INTO с предложением VALUES для каждой добавляемой новой записи.

Обновление данных

Инструкция **UPDATE** создает запрос на обновление, который изменяет значения полей указанной таблицы на основе заданного условия отбора. Синтаксис команды:

UPDATE таблица
SET новое Значение
WHERE условие Отбора;

где таблица — имя таблицы, данные в которой следует изменить;

новое Значение — выражение, определяющее значение, которое должно быть вставлено в указанное поле обновленных записей;

условие Отбора — выражение, отбирающее записи, которые должны быть изменены.

При выполнении этой инструкции будут изменены только записи, удовлетворяющие указанному условию. Инструкцию UPDATE особенно удобно использовать для изменения сразу нескольких записей или в том случае, если записи, подлежащие изменению, находятся в разных таблицах. Одновременно можно изменить значения нескольких полей. Следующая инструкция SQL увеличивает стипендию студентов группы 422-1 на 10 процентов:

```
UPDATE Студенты
SET Стипендия = стипендия * 1.1
WHERE Номер_группы = '422-1';
```

Запрос на объединение

Операция **UNION** создает запрос на объединение, который объединяет результаты нескольких независимых запросов или таблиц. Синтаксис команды: [TABLE] запрос_1 UNION [ALL] [TABLE] запрос_2 [UNION[ALL] [TABLE] запрос_n [...]]

где запрос_1-n — инструкция SELECT или имя сохраненной таблицы, перед которым стоит зарезервированное слово TABLE.

В одной операции UNION можно объединить в любом наборе результаты нескольких запросов, таблиц и инструкций SELECT. В следующем примере объединяется существующая таблица «Студенты» и инструкции SELECT:

```
TABLE Студенты UNION ALL
SELECT *
FROM Абитуриеты
WHERE Общий_балл > 22;
```

По умолчанию повторяющиеся записи не возвращаются при использовании операции UNION, однако в нее можно добавить предикат

ALL, чтобы гарантировать возврат всех записей. Кроме того, такие запросы выполняются несколько быстрее.

Все запросы, включенные в операцию UNION, должны отбирать одинаковое число полей; при этом типы данных и размеры полей не обязаны совпадать. Псевдонимы необходимо использовать только в первом предложении SELECT, в остальных они пропускаются.

В каждом аргументе «Запрос» допускается использование предложения GROUP BY или HAVING для группировки возвращаемых данных. В конец последнего аргумента «Запрос» можно включить предложение ORDER BY, чтобы отсортировать возвращенные данные.

7.1.4. Основные различия Microsoft Jet SQL и ANSI SQL

Рассмотрим различия двух диалектов языка SQL Microsoft Jet SQL и ANSI SQL, описанные в руководстве разработчика приложений баз данных на СУБД Microsoft Access:

– языки SQL ядра базы данных Microsoft Jet и ANSI SQL имеют разные наборы зарезервированных слов и типов данных;

– разные правила применимы к оператору Between...And, имеющему следующий синтаксис: выражение [NOT] Between значение_1 And значение_2. В языке SQL Microsoft Jet *значение_1* может превышать *значение_2*; в ANSI SQL, *значение_1* должно быть меньше или равно *значение_2*;

– разные подстановочные знаки используются с оператором Like. Так, в языке SQL Microsoft Jet любой одиночный символ изображается знаком «?», а в ANSI SQL знаком «_», любое число символов в языке SQL Microsoft Jet изображается знаком «*», а в ANSI SQL знаком «%»;

– язык SQL ядра Microsoft Jet обычно предоставляет пользователю большую свободу. Например, разрешается группировка и сортировка по выражениям.

7.1.5. Особые средства языка SQL Microsoft Jet

В языке SQL Microsoft Jet поддерживаются следующие дополнительные средства:

– инструкция TRANSFORM предназначенная для создания перекрестных запросов;

– дополнительные статистические функции, такие как StDev и VarP;

– описание PARAMETERS, предназначенное для создания запросов с параметрами.

7.1.6. Средства ANSI SQL, не поддерживаемые в языке SQL Microsoft Jet

В языке SQL Microsoft Jet не поддерживаются следующие средства ANSI SQL:

– инструкции, управляющие защитой, такие, как COMMIT, GRANT и LOCK;

– зарезервированное слово DISTINCT в качестве описания аргумента статистической функции. Например, в языке SQL Microsoft Jet нельзя использовать выражение SUM(DISTINCT имя Столбца);

– предложение LIMIT TO nn ROWS, используемое для ограничения количества строк, возвращаемых в результате выполнения запроса. Для ограничения количества возвращаемых запросом строк можно использовать только предложение WHERE.

7.2. Язык Query-by-Example

7.2.1. Основы QBE

Основной принцип формирования запросов на языке QBE заключается в том, что запрос на обработку формулируется путем заполнения некоторой пустой таблицы, в основном соответствующей исходному отношению, являющемуся источником записей для результирующего набора записей. В таблицу добавляются дополнительные столбцы, если в результирующем наборе данных необходимо наличие столбца, значение которого является константой, либо является результатом вычисления на основе значений других столбцов таблицы-источника запроса.

Лидирующий столбец таблицы-запроса согласно терминологии, предложенной Дж. Ульманом [17], в заголовке содержит имя источника запроса, а в теле наименование операций манипулирования данными.

Остальные столбцы таблицы-запроса в заголовке содержат имена атрибутов отношения, а в строках содержатся элементы запроса, относящиеся к соответствующим атрибутам — различные параметры, значения и критерии запроса.

При описании команд QBE будем использовать терминологию, предложенную Дж. Ульманом, а затем рассмотрим принципы применения запросов по образцу в среде MS Access. Команда выборки данных, обозначается символом «P.». Наличие этого оператора в первом столбце говорит о том, что все атрибуты отношения будут представле-

ны в результирующем наборе данных. При необходимости обеспечить выборку определенных атрибутов отношения, оператор «P.» отображается в соответствующих столбцах, содержащих в заголовке имена этих атрибутов. Для задания условий отбора в столбце соответствующего атрибута указывается критерий отбора и один из знаков сравнения. Также в состав команд языка QBE включены команды: добавления данных I. — insert.; обновления данных U. — update.; удаления D. — delete.

На рис. 7.7 представлены примеры запросов, созданные в идеологии QBE. Так, в результате выполнения первого запроса будут выданы сведения о студентах группы 422-1. Результатом второго запроса является набор данных с одним атрибутом ФИО СТУДЕНТА, значением которого будут студенты, родившиеся в г. Чита. Третий запрос добавит в отношение СТУДЕНТЫ новую запись. Четвертый запрос изменит фамилию студентки с № зачетной книжки 1992432-02. Наконец, в результате выполнения последнего запроса будут удалены записи содержащие сведения о студентах группы 422-3.

Студенты	№ зачетной книжки	ФИО студента	Дата рождения	Место рождения	№ Группы
P.					= 422-1
		P.		= Чита	
I.	1992432-13	Сидоров А.Н.	05.09.1985	Омск	422-1
U.	1992432-02	Казакова А.Н.			
D.					422-3

Рис. 7.7. Примеры записи запросов средствами QBE

7.2.2. Запрос по образцу (идеология MS ACCESS)

В СУБД MS Access существует специальное средство построения запросов, которое более наглядно позволяет пояснить принцип построения запросов в терминах QBE. Так, нижняя часть построителя запросов в MS Access является бланком запроса MS Access, или как его называют областью QBE (рис. 7.8.).

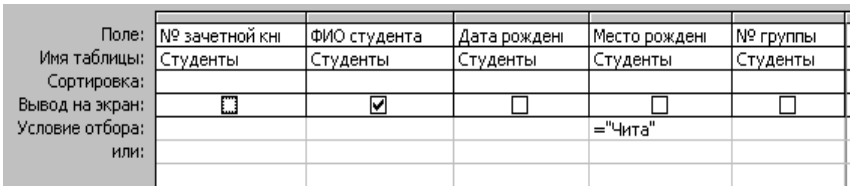

Рис. 7.8. Пример записи запроса в бланке построения запросов MS Access

Здесь указываются параметры запроса и данные, которые нужно отобрать (аналог перечня условий предложения WHERE в SQL-запросах), а также определяется способ их отображения на экране. В запрос не обязательно включать все поля выбранных таблиц.

В общем случае поля, вводимые в наборе записей запроса, наследуют свойства, заданные для соответствующих полей таблицы.

При создании запроса можно задать критерии, вследствие чего по запросу будет осуществлен отбор только нужных записей.

Чтобы найти записи с конкретным значением в каком либо поле, нужно ввести это значение в данное поле в строке бланка QBE «Условие отбора». Нечисловые критерии, устанавливаемые в QBE-области, должны быть заключены в кавычки.

Для создания запроса с несколькими критериями пользуются различными операторами.

Можно задать несколько условий отбора, соединенных логическим оператором или (or), для некоторого поля одним из двух способов:

1) можно ввести все условия в одну ячейку строки «Условие отбора», соединив их логическим оператором *или* (*or*);

2) ввести второе условие в отдельную ячейку строки *или* (*or*). Если используется несколько строк *или* (*or*), то, чтобы запись была выбрана, достаточно выполнения условий хотя бы в одной из строк.

Логическая операция *и* (*and*) используется в том случае, когда должны быть выполнены оба условия, и только в этом случае запись будет выбрана.

Оператор Between позволяет задать диапазон значений, например: between 10 and 20.

Оператор In позволяет задавать используемый для сравнения список значений. Например: in (“первый”,”второй”,”третий”).

Оператор Like полезен для поиска образцов в текстовых полях, причем можно использовать шаблоны:

* обозначает любое количество (включая нулевой) символов;

? — любой одиночный символ;

указывает, что в данной позиции должна быть цифра.

Например здесь, также как и в тексте SQL-запроса на выборку, для выбора ФИО студента, начинающейся с буквы «С» и с окончанием «о» можно записать like С*о

Можно ввести дату и время, при этом значения должны быть заключены между символами #. Например: #10 мая 1998#

Также в бланке построения запросов СУБД MS Access используется ряд других функций, которые помогут задать условия отбора для даты и времени, например:

Day(дата) возвращает значение дня месяца в диапазоне от 1 до 31;

Month(дата) возвращает значение месяца года в диапазоне от 1 до 12;

Year(дата) возвращает значение года в диапазоне от 100 до 9999.

Можно задать вычисления над любыми полями таблицы и сделать вычисляемое значение новым полем в запросе. Для этого в строке «Поле» бланка QBE вводится формула для вычисления, причем имена полей заключаются в квадратные скобки.

Например: =[Оклад]*0.15.

В выражениях можно использовать следующие операторы:

арифметические: * + - / ^;

соединение частей текста при помощи знака & , например:

=[Фамилия] & “ “&[Имя].

Итоговые запросы значительно отличаются от обычных. В них поля делятся на 2 типа:

поля, по которым осуществляется группировка данных;

поля, для которых проводятся вычисления.

Access предоставляет ряд функций, обеспечивающих выполнение групповых операций.

Основные групповые функции, которыми можно воспользоваться:

SUM — вычисление суммы всех значений заданного поля (для числовых или денежных полей), отобранных запросом;

AVG — вычисление среднего значения в тех записях определенного поля, которые отобраны запросом (для числовых или денежных полей);

MIN — выбор минимального значения в записях определенного поля, отобранных запросом;

MAX — выбор максимального значения в записях определенного поля, отобранных запросом;

COUNT — вычисление количества записей, отобранных запросом в определенном поле, в которых значения данного поля отличны от нуля;

FIRST — определение первого значения в указанном поле записей;

LAST — определение последнего значения в указанном поле записей.

Готовый запрос на выборку записей, созданный с помощью бланка QBE, выполняется, в результате чего получается таблица с ответом на заданные условия.

Запросы к нескольким таблицам.

Запросы можно создавать для отбора данных как из одной, так и из нескольких таблиц. Запросы к нескольким таблицам производятся

аналогично запросам к однотабличным БД с той лишь разницей, что в окно конструктора запроса добавляются все таблицы, данные которых нужны в запросе.

При этом следует учитывать наличие связей между таблицами.

Помимо запросов на выборку, в MS Access с помощью бланка построения запросов можно создавать запросы на изменение, добавление, удаление данных и перекрестные запросы. Перекрестный запрос, результат выполнения которого представлен на рис. 7.4, может также быть реализован с помощью бланка построения запросов (рис. 7.10.)

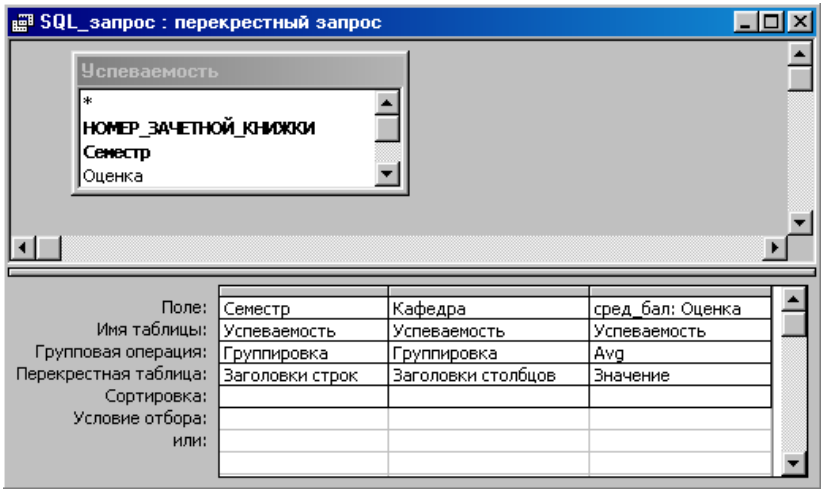

Рис. 7.10. Перекрестный запрос, созданный с помощью бланка построителей запросов

Следует отметить, что любой запрос, созданный с помощью построителя запросов в среде MS Access преобразуется в SQL-запрос.

Контрольные вопросы

1. Поясните необходимость наличия средств манипулирования данными в СУБД.
2. Какие операции с данными можно осуществлять с помощью средств языка SQL и нельзя средствами QBE.
3. Создайте несколько SQL-запросов, используя операции выборки, обновления и удаления данных.

8. ФИЗИЧЕСКАЯ СТРУКТУРА ДАННЫХ

8.1. Структуры внешней памяти, методы организации индексов

8.1.1. Организация внешней памяти

Знание физической структуры данных позволяет обеспечить качественное выполнение физического проектирования БД. Физическое проектирование БД — это отдельный процесс, тесно связанный с логическим проектированием, а также процесс организации хранения данных с проектированием формата хранимой записи, классификации записей и с управлением размещения наборов данных.

Реляционные СУБД обладают рядом особенностей, влияющих на организацию внешней памяти. К наиболее важным особенностям можно отнести следующие [1]:

1) наличие двух уровней системы: уровня непосредственного управления данными во внешней памяти (а также обычно управления буферами оперативной памяти, управления транзакциями и журнализацией изменений БД) и языкового уровня (например, уровня, реализующего язык SQL). При такой организации подсистема нижнего уровня должна поддерживать во внешней памяти набор базовых структур, конкретная интерпретация которых входит в число функций подсистемы верхнего уровня;

2) поддержание отношений-каталогов. Информация, связанная с именованием объектов базы данных и их конкретными свойствами (например, структура ключа индекса), поддерживается подсистемой языкового уровня. С точки зрения структур внешней памяти отношение-каталог ничем не отличается от обычного отношения базы данных;

3) регулярность структур данных. Поскольку основным объектом реляционной модели данных является плоская таблица, главный набор объектов внешней памяти может иметь очень простую регулярную структуру. При этом необходимо обеспечить возможность эффективного выполнения операторов языкового уровня как над одним отношением (простые селекция и проекция), так и над несколькими отношениями (наиболее распространено и трудоемко соединение нескольких отношений). Для этого во внешней памяти должны поддерживаться дополнительные «управляющие» структуры — индексы;

4) наконец, для выполнения требования надежного хранения баз данных необходимо поддерживать избыточность хранения данных, что обычно реализуется в виде журнала изменений базы данных.

Соответственно возникают следующие разновидности объектов во внешней памяти базы данных [1]:

– ***строки отношений*** — основная часть базы данных, большей частью непосредственно видимая пользователям;

– ***управляющие структуры*** — индексы, создаваемые по инициативе пользователя (администратора) или верхнего уровня системы из соображений повышения эффективности выполнения запросов и обычно автоматически поддерживаемые нижним уровнем системы;

– ***журнальная информация***, поддерживаемая для удовлетворения потребности в надежном хранении данных;

– ***служебная информация,*** поддерживаемая для удовлетворения внутренних потребностей нижнего уровня системы (например, информация о свободной памяти).

Любая конкретная СУБД основывается на конкретном комплексном решении задач связанных с организацией хранения и управления данными. Мы рассмотрим лишь фрагменты таких решений (эскизы).

8.1.2. Хранение отношений в базе данных

Существуют два возможных способа физического хранения отношений: покортежное хранение отношений и хранение отношений по столбцам. Покортежное хранение, при котором кортеж является физической единицей хранения, обеспечивает быстрый доступ к кортежу целиком, но замедляется работа с БД при необходимости оперировать только частью кортежа. Хранение отношения по столбцам, при котором единицей хранения является атрибут отношения. В этом случае в среднем тратится меньше памяти, необходимой для хранения отношения, т.к. исключаются дублирующие значения атрибутов, однако при такой организации хранения необходимо наличие дополнительных надстроек, обеспечивающих связь разрозненно хранящихся значений атрибутов в единый кортеж отношения.

В каждой конкретной СУБД существует свой формат хранения отношений. Наиболее открытым с точки зрения визуального представления является формат DBF, используемый в СУБД семейства dBase (dBase III,IV,V, FoxPro 2.x), в котором применяется покортежное хранение отношений (структура формата описана в литературе по работе с СУБД FoxPro 2.x)

8.1.3. Методы доступа к данным и способы организации индексов

Во всех существующих на рынке СУБД имеется в наличии средство, обеспечивающее оптимальный по скорости доступ к данным. Такая надстройка над данными называется индексами базы данных. В целом индекс можно описать как специальную структура данных, создаваемая автоматически или по запросу пользователя. Работа с Индексом выглядит также, как и с предметным указателем. СУБД все делает автоматически, при этом в БД для формирования индекса может быть использован любой атрибут отношения, в том числе и составной. В индексе значения атрибута хранятся упорядочено, каждому значению соответствует указатель на строку отношения, которое его содержит (аналог номера страницы в предметном указателе).

Индекс описывает отношения упорядочивания и однозначности значений, с помощью которых обеспечивается эффективный доступ к записям в таблицах базы данных.

При этом следует отметить, что как бы ни были организованы индексы, их назначение состоит в обеспечении эффективного доступа к кортежу отношения по некоторому ключу. Ключом индекса является значение атрибута отношения. Индекс может быть построен на любом атрибуте отношения. Первичный ключ отношения практически всегда является индексированным полем таблицы. Такой индекс является уникальным по определению.

Важным свойством индекса является обеспечение сортировки данных в отношении, что позволяет осуществлять последовательный просмотр кортежей отношения в порядке возрастания или убывания значений ключа.

Общей идеей любой организации индекса, поддерживающего прямой доступ по ключу и последовательный просмотр в порядке возрастания или убывания значений ключа, является хранение упорядоченного списка значений ключа с привязкой к каждому значению ключа списка идентификаторов кортежей. Одна организация индекса отличается от другой главным образом в способе поиска ключа с заданным значением [1].

Различают следующие методы хранения и доступа к данным: физический последовательный, индексно-последовательный, индексно-произвольный, инвертированный, метод хеширования.

Опишем и охарактеризуем представленные методы, исходя из следующих критериев: эффективности доступа — величины обратной среднему числу обращений, необходимых для осуществления запроса

конкретной записи БД; эффективности хранения — величины, обратной среднему числу байтов памяти, требуемого для хранения одного байта исходных данных согласно [7].

Физический последовательный

Значения ключей физических записей находятся в логической последовательности. Применяется в основном для дампа и восстановления данных. Может применяться как для хранения, так и для выборки данных. Эффективность использования памяти близка к 100 %. Эффективность доступа физического последовательного метода оставляет желать лучшего, поскольку для выборки нужной записи требуется просмотреть все предыдущие ей записи БД.

Индексно-последовательный

Метод доступа, при использовании которого до осуществления доступа к собственно записям БД проверяются значения ключей, называется индексно-последовательным. Значения ключей физических записей находится в логической последовательности. Может применяться как для хранения, так и для выборки данных. В индекс значений ключей заносятся статьи значений ключей в блоках. Наличие дубликатов значений ключей недопустимо. Эффективность доступа зависит от числа уровней индексации, распределения памяти для размещения индекса, числа записей базы данных и уровня переполнения. Эффективность хранения зависит от размера и изменяемости базы данных.

Индексно-произвольный

При индексно-произвольном методе доступа записи хранятся в произвольном порядке. Создается отдельный файл либо раздел файла БД (в зависимости от СУБД) из статей, содержащих значения действительного ключа и физические адреса хранимых записей.

Значения ключей физических записей необязательно находятся в логической последовательности. Хранение и доступ к индексу могут осуществляться с помощью индексно-последовательного метода доступа. Индекс содержит статью для каждой записи БД. Эти статьи упорядочены по возрастанию. Ключи индекса сохраняют логическую последовательность. Записи БД могут быть не упорядочены по возрастанию ключа. Метод может использоваться как для запоминания, так и для выборки данных.

Метод прямого доступа

Не требует упорядоченности значений ключей физических записей. Между ключом записи и ее физическим адресом существует взаимно однозначное соответствие. Метод может применяться как для хранения, так и для поиска записей. Для поиска одной записи используется одно обращение к индексу. Эффективность хранения зависит от плотности ключей. Наличие дубликатов ключей недопустимо.

Метод доступа посредством хеширования

Не требуется логическая упорядоченность значений ключей физических записей. Значениям нескольких ключей может соответствовать один и тот же физический адрес (блок). Может применяться как для хранения, так и для поиска записей. Эффективность доступа и хранения зависят от распределения ключей, алгоритма их преобразования и распределения памяти. Между методом прямого доступа и методом доступа посредством хеширования существует сходство. При методе доступа посредством хеширования адрес физической записи алгоритмически определяется из значения ключа записи.

Инвертированный (метод вторичного индексирования)

Значения ключей физических записей необязательно находятся в логической последовательности. Метод применяется только для выборки данных. Индекс может быть построен для каждого инвертированного поля. Эффективность доступа зависит от числа записей БД, числа уровней индексации и распределения памяти для размещения индекса.

Инвертированные индексы (также называемые словарными файлами) – представляют собой список выбранных из линейного файла, поисковых слов или фраз, помещенных в отдельный, организованный в алфавитном порядке файл, со ссылками на определенную часть записи в линейном файле.

Инвертированные списки формируются системой для поисковых атрибутов, причем для каждого возможного значения такого атрибута составляется список уникальных номеров записей, в которых это значение атрибутов присутствует. Записи с одним и тем же значением поля группируются, а общее для всей группы значение используется в качестве указателя этой группы. Тогда при поиске записей по значениям поисковых атрибутов системе достаточно отыскать списки, соответствующие требуемым значениям, и выбрать номер записи согласно заданной «схеме» пересечения или объединения условий на значениях

поисковых атрибутов, а также отрицания некоторого условия. На рис. 8.1. приведен пример поиска записей инверсированным методом доступа.

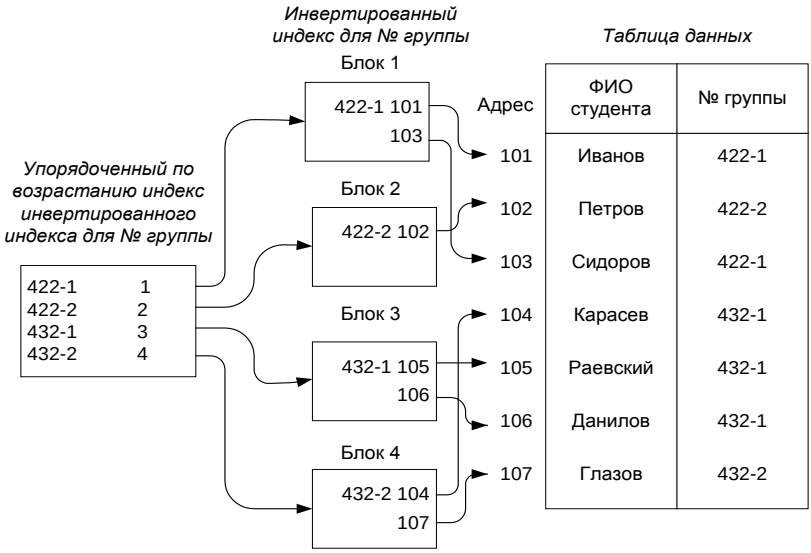

Рис. 8.1. Инвертированный метод доступа

Рассмотрим более подробно прямой метод доступа и метод доступа посредством хеширования.

Прямой метод доступа

Основная особенность прямого метода доступа заключается во взаимнооднозначном соответствии между ключом записи и ее физическим адресом. Физическое местоположение записи определяется непосредственно из значения ключа.

Эффективность доступа всегда равна единице, т.е. доступ к записи осуществляется за одно обращение к таблице, поскольку в данном случае речь идет о необходимости наличия уникального индекса.

Эффективность хранения зависит от плотности ключей. При низкой плотности память расходуется впустую, поскольку резервируются адреса, соответствующие отсутствующим ключам. В ряде случаев не требуется однозначное соответствие между ключом и физическим адресом; записи вполне достаточно, чтобы группа ключей ссылалась на

один и тот же физический адрес. Такой метод доступа называется методом доступа посредством хеширования.

Метод хеширования

Метод хеширования — разновидность метода прямого доступа, обеспечивающего быструю выборку и обновление записей.

Общей идеей методов хеширования является применение к значению ключа некоторой функции свертки (функции хеширования), вырабатывающей значение меньшего размера. Свертка значения ключа затем используется для доступа к записи.

Хешированием называется метод доступа, обеспечивающий прямую адресацию данных путем преобразования значения ключа в относительный или абсолютный физический адрес. Алгоритм преобразования ключа называют также подпрограммой рандомизации. При использовании функции хеширования возможно преобразование двух или более ключей в один и тот же физический адрес, который называется собственным адресом. Записи, ключи которых отображаются в один и тот же физический адрес, называются синонимами, а случай преобразования нового ключа в уже заданный собственный адрес называется коллизией. Поскольку по адресу, определяемому функцией хеширования, может физически храниться только одна запись, синонимы должны храниться в каких-нибудь других ячейках памяти. При возникновении коллизий образуются цепочки синонимов, необходимых для обеспечения механизма поиска синонимов (рис. 8.2).

Сущность метода хеширования заключается в том, что все адресное пространство делится на несколько областей фиксированного размера, которые называются ***бакетами***. В качестве бакета может выступать цилиндр, дорожка, блок, страница, т.е. любой участок памяти, адресуемый в операционной среде как единое целое. Наименьшая составная единицей бакета называется фрагментом записи или секцией. Если при занесении нового значения индекса все бакеты заняты, то для него выделяется дополнительная область памяти, называемая областью переполнения.

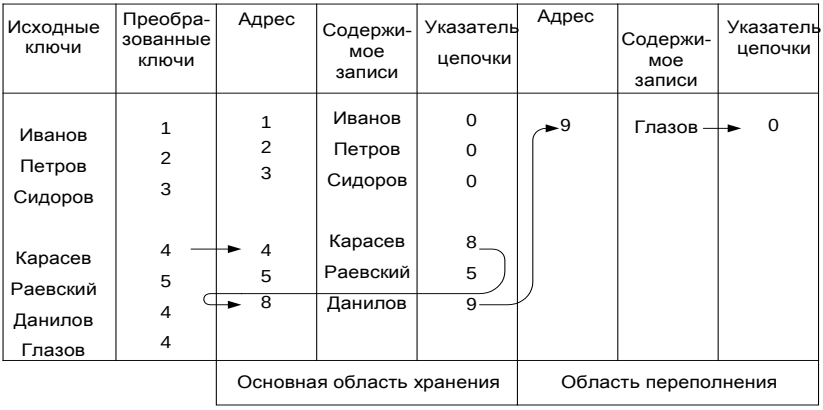

Рис. 8.2. Пример цепочки синонимов

Главным ограничением этого метода является фиксированный размер таблицы. Если таблица заполнена слишком сильно или переполнена, то возникнет слишком много цепочек переполнения, и главное преимущество хеширования — доступ к записи почти всегда за одно обращение к таблице — будет утрачено. Расширение таблицы требует ее полной переделки на основе новой хэш-функции (со значением свертки большего размера).

В случае баз данных такие действия являются абсолютно неприемлемыми. Поэтому обычно вводят промежуточные таблицы-справочники, содержащие значения ключей и адреса записей, а сами записи хранятся отдельно. Тогда при переполнении справочника требуется только его переделка, что вызывает меньше накладных расходов.

Характеристики метода:

1) при хеш-файлах распределение ключевых значений оказывает значительное влияние на распределение собственных адресов и количество синонимов;

2) вид функции хеширования оказывает влияние на распределение собственных адресов и на количество синонимов;

3) упорядоченность данных при загрузке данных влияет на общую производительность системы;

4) объем адресного пространства или количество бакетов влияет на количество синонимов и коэффициент резервирования памяти;

5) большой размер бакета обеспечивает гибкость обработки коллизий без использования области переполнения;

6) метод обработки переполнения влияет на время обслуживания и зависит от метода хеширования ключа;

7) хеширование ключа обеспечивает эффективную выборку и обновление записей. Платой за эту эффективность является нарушение логической упорядоченности файла;

8) хеширование допускает возможность вторичного преобразования по специальному ключу.

Новейшие типы индексов

Двоичный масочный индекс (bitmap)

В индексе этого типа двоичная маска формируется на основе значений, допустимых для столбца индексируемой таблицы, с учетом их действительных значений, уже внесенных в таблицу. Бит устанавливается равным единице (1), если соответствующее значение из набора допустимых совпадает со значением в данной строке таблицы. В противном случае биту присваивается значение ноль (0). Если набор допустимых значений в столбце невелик, то такой масочный индекс оказывается более компактным и обрабатывается быстрее, чем классические индексы [18]. Пример двоичного индекса представлен на рис. 8.3.

Страна проживания студента	**Маска**
Россия	1 0 1 0 0 0 1 0
Казахстан	0 1 0 1 0 0 0 0
Кыргызстан	0 0 0 0 1 0 0 0
Украина	0 0 0 0 0 1 0 0
Беларусь	0 0 0 0 0 0 0 1

Рис. 8.3. Масочный индекс для таблицы студенты

В данном примере представлены двоичные маски для таблицы СТУДЕНТЫ. Ключом индекса является столбец СТРАНА ПРОЖИВАНИЯ СТУДЕНТА. В таблице СТУДЕНТЫ 20 строк. В маске каждый бит соответствует одной записи и устанавливается равным 1, если значение атрибута Country (СТРАНА ПРОЖИВАНИЯ СТУДЕНТА) совпадает со значением параметра для этой маски. В таблице СТУДЕНТЫ, в строках 1, 3, 7 указана страна проживания Россия, в строке 8 – Беларусь и т.д. Такой метод индексирования широко применяется в СУБД Oracle 8i.

Кластерный индекс

Следует отметить наличие в некоторых СУБД кластеров таблиц. Кластеры таблиц – это объект БД, который физически группирует

совместно используемые таблицы в пределах одного блока. Кластеризация таблиц дает значительный эффект в том случае, если в системе приходится оперировать запросами, которые требуют совместной обработки данных из нескольких таблиц. В кластере таблицы хранятся ключ кластера (столбец, который используется для объединения таблиц) и значения из столбцов в кластеризованных таблицах. Поскольку кластеризованные таблицы хранятся в одном блоке БД, время на выполнение операций ввода-вывода заметно сокращается [18].

В некоторых СУБД по идеологии кластеров таблиц строится кластерный индекс, который использует столбец (ключ кластера). В отличие от обычного индекса в кластерном индексе хранится значение ключа только один раз независимо от того, сколько раз ключ встречается в таблице.

Представляет интерес реализация механизма кластерного индекса для некоторых форматов настольных СУБД Paradox, dBase. В этих СУБД кластерный индекс состоит из одного или нескольких неключевых полей, которые, взятые вместе, позволяют расположить все записи таблицы в заранее определенном порядке. При этом кластерный индекс предоставляет возможность эффективного доступа к записям таблицы, в которой значения индекса могут не быть уникальными.

8.1.4. Управление индексами

Поскольку организация индексов в большинстве СУБД является довольно сложной, управление индексами требует повышенного внимания. Зачастую с целью улучшения производительности БД программисты стараются увеличить количество индексов. Однако увеличение количества индексов может привести к обратному эффекту и значительно ухудшить производительность в операциях обновления. В связи с этим необходимо следить за индексами и удалять те из них, которые не используются. В некоторых СУБД существует специальная функция, позволяющая выяснить, даст ли добавление индекса желаемый эффект.

8.1.5. Словарь данных

Словарь данных — это хранилище информации обо всех объектах, входящих в состав БД. СУБД использует словарь для получения информации об объектах и ограничениях прав доступа к ним. Конкретные пользователи и администратор БД могут получить из словаря

интересующую информацию о БД, а именно — информацию о таблицах, индексах, представлениях, пакетах и процедурах [18].

Например, словарь данных СУБД ORACLE может предоставлять следующую информацию:

– имена пользователей ORACLE

– привилегии и роли, которые были предоставлены каждому пользователю;

– имена объектов схем (таблиц, представлений, снимков, индексов, кластеров, синонимов, последовательностей, процедур, функций, пакетов, триггеров и т.д.);

– информацию об ограничениях целостности;

– умалчиваемые значения для столбцов;

– сколько пространства было распределено и в настоящее время используется объектами в базе данных;

– информацию аудита, например, кто обращался к различным объектам и обновлял их другую общую информацию о базе данных.

Доступ к словарю данных возможен только в режиме чтения.

Одной из составляющих словаря данных и ключевым компонентом всей информационной структуры, являются *внутренние таблицы* СУБД. К ним обращается система за всей внутренней информацией о текущем состоянии и процессах, происходящих в системе. Имена этих таблиц зашифрованы и в некоторых СУБД скрыты от пользователя, а в большинстве своем структура их не документирована, так что предпринять осмысленные действия с системной таблицей даже администратору БД проблематично. Однако в них хранится множество сведений и данных о конфигурации системы.

Словарь данных призван помочь пользователю в выполнении следующих функций [7]:

– установлении связи с другими пользователями;

– осуществлении простого и эффективного управления элементами данных при вводе в систему новых элементов или изменения описания существующих;

– уменьшении избыточности и противоречивости данных;

– определении степени влияния изменений в элементах данных на всю БД;

– централизации управления элементами данных с целью упрощения проектирования БД и расширения.

Существует понятие идеального словаря данных, словаря, обеспечивающего полноценную связь между всеми уровнями моделей БД и объектами БД.

Такой словарь должен [7]:

1) поддерживать концептуальную, физическую и внешние модели данных;

2) быть интегрированным в СУБД;

3) должен поддерживать возможность хранения нескольких версий программной реализации;

4) обеспечивать эффективный обмен информацией внешних программ с СУБД (в идеале привязка внешних и внутренних моделей должна происходить во время выполнения программ, использующих БД, при этом должно осуществляться динамическое построение описания БД).

8.1.6. Прочие объекты БД

Представления (views) — это хранимые предложения SQL, которые можно запросить. Они используются из соображений распределения предоставляемых пользователю определенных данных. С помощью представлений возможно упростить сложные запросы и сделать их более понятными, а также появляется возможность скрыть распределенные объекты БД. Любое выражение, представленное в виде SQL-запроса, можно оформить в виде представления. Наибольшая польза от представлений становится заметной при разработке приложений, поскольку они дают возможность скрыть структуру запроса и вместо него использовать простой синтаксис обращения к представлению как к таблице БД. При формировании представлений можно оптимизировать структуру промежуточной таблицы и таким образом обеспечить высокую производительность системы в ходе выполнения запроса. Большинство современных СУБД, использующих представления, трактуют их как виртуальные таблицы: везде, где применяется таблица, в SQL-запросах на выборку данных ее можно заменить представлением. Однако данные в представлении никогда не сохраняются, они всегда создаются при открытии представления.

Триггеры (triggers) — это хранимые процедуры, которые запускаются при выполнении определенных действий с таблицей. Можно создать триггеры, которые будут запускаться при операциях вставки, удаления или обновления данных. Возможны варианты создания таких триггеров, которые будут выполняться при обращении к каждой строке или при каждом запросе к таблице. Триггеры представляют удобное средство для обеспечения логической целостности данных.

Хранимые пакеты, процедуры и функции

Хранимые пакеты, процедуры и функции хранятся в *словаре данных*. Там же сохраняется их исходный код. Хранимая процедура — это

выполняемый объект, которому можно передать аргументы и получать от него сформированные результаты. Хранимая функция отличается от хранимой процедуры тем, что возвращаемым результатом выполнения функции является некоторое единичное значение. Пакет представляет собой совокупность процедур, переменных и функций, объединенных для выполнения некоторой задачи. Следует отметить, что принципы реализации хранимых процедур различаются для каждой конкретной СУБД, однако в основе всех принципов лежит использование процедурного расширения языка SQL. Хранимые процедуры и функции могут также содержать аргументы ввода, необходимые для формирования динамического запроса. Хранимые процедуры и функции могут определятся относительно одной или нескольких таблиц БД [18].

Последовательности — это объекты БД, введенные в некоторые СУБД нового поколения (Oracle, MS Access 97 и др.), которые используются для формирования уникальных числовых величин. При каждом извлечении очередного числа из последовательности происходит его приращение. Последовательности используются при формировании уникальных чисел для конкретного поля таблицы, являющегося первичным ключом.

Журнальная информация

Журнал обычно представляет собой чисто последовательный файл с записями переменного размера, которые можно просматривать в прямом или обратном порядке. Обмены производятся стандартными порциями (страницами) с использованием буфера оперативной памяти. Структура журнальных записей известна только компонентам СУБД, ответственным за журнализацию и восстановление. Поскольку содержимое журнала является критичным при восстановлении базы данных после сбоев, к ведению файла журнала предъявляются особые требования по части надежности [1]. Обычно поддерживается две копии журнала на разных дисках.

Возможны два основных варианта ведения журнальной информации. В первом варианте для каждой транзакции поддерживается отдельный локальный журнал изменений базы данных этой транзакцией. Эти локальные журналы используются для индивидуальных откатов транзакций и могут поддерживаться в оперативной (правильнее сказать, в виртуальной) памяти. Кроме того, поддерживается общий журнал изменений базы данных, используемый для восстановления состояния базы данных после мягких и жестких сбоев.

Этот подход позволяет быстро выполнять индивидуальные откаты транзакций, но приводит к дублированию информации в локальных и общем журналах. Поэтому чаще используется второй вариант - под-

держание только общего журнала изменений базы данных, который используется и при выполнении индивидуальных откатов

Служебная информация

Для корректной работы подсистемы управления данными во внешней памяти необходимо поддерживать информацию, которая используется только этой подсистемой и не видна подсистеме языкового уровня. Набор структур служебной информации зависит от общей организации системы, но обычно требуется поддержание следующих служебных данных [1]:

– внутренних каталогов, описывающих физические свойства объектов базы данных, например, число столбцов таблицы, их размер, типы данных; описание индексов, определенных для данной таблицы;

– описателей свободной и занятой памяти в страницах отношения. Такая информация требуется для нахождения свободного места при занесении новой записи в таблицу;

– связывания страниц одного отношения. Если в одном файле располагаются страницы нескольких отношений, то нужно каким-то образом связать страницы одного отношения. В этом случае стараются использовать косвенное связывание страниц с использованием служебных индексов.

8.2. Оптимизация работы с БД

8.2.1. Оптимизация работы с таблицами

Существует несколько возможностей оптимизировать работу с таблицами.

1. Необходимо создавать таблицы, не содержащие избыточных данных.

2. Индексы следует создавать для сортируемых и объединяемых полей, а также для полей, используемых при задании условий отбора в SQL-запросах. Повысить быстродействие при выполнении SQL-запросов можно также проиндексировав поля, являющиеся связующими полями для нескольких таблиц (внешние ключи).

3. При определении типа поля необходимо выбирать подходящий тип данных. Это поможет уменьшить размеры базы данных и увеличит скорость выполнения операций связи. При описании поля следует задать для него тип данных наименьшего размера, позволяющий хранить нужные данные.

При выборе типа данных, используемых в поле, необходимо учитывать следующее:

– тип значений, которые должны отображаться в поле (например, нельзя хранить текст в поле, имеющем числовой тип данных);

– размер данных для хранения значений в поле;

– возможность применения математических и других операций со значениями в поле (например, суммировать значения можно в числовых полях и в полях, имеющих валютный формат, а значения в текстовых полях и полях объектов OLE нельзя);

– необходимость сортировки или индексирования поля (сортировать и индексировать поля MEMO, гиперссылки и объекты OLE невозможно);

– необходимость использования полей в группировке записей в запросах или отчетах. Поля MEMO, гиперссылки и объекты OLE использовать для группировки записей нельзя;

– порядок сортировки значений в поле. Числа в текстовых полях сортируются как строки чисел (1, 10, 100, 2, 20, 200 и т.д.), а не как числовые значения. Для сортировки чисел как числовых значений необходимо использовать числовые поля или поля, имеющие денежный формат (если СУБД поддерживает такой тип данных). Также многие форматы дат невозможно отсортировать надлежащим образом, если они были введены в текстовое поле.

Поля с типом данных объект OLE используются для хранения таких данных, как документы Microsoft Word или Microsoft Excel, рисунки, звук и объекты других программ. Объекты OLE могут быть связаны или внедрены в поля таблиц СУБД нового поколения, поддерживающих возможность работы с OLE-объектами.

8.2.2. Ограничения целостности

В разделе *4.2.4* мы подробно рассматривали характеристики реляционных отношений. Здесь же отметим еще одно определение целостности и опишем некоторые важные моменты, связанные с этим понятием. Будем считать, что целостность понимается как правильность данных в любой момент времени. Но эта цель может быть достигнута лишь в определенных пределах: СУБД не может контролировать правильность каждого отдельного значения, вводимого в базу данных. Например, нельзя обнаружить, что вводимое значение 5 (представляющее номер дня недели) в действительности должно быть равно 3. С другой стороны, значение 9 явно будет ошибочным и СУБД должна его отвергнуть. Однако для этого ей следует сообщить, что номера дней недели должны принадлежать набору (1, 2, 3, 4, 5, 6, 7).

Поддержание целостности базы данных может рассматриваться как защита данных от неверных изменений или разрушений. Современные СУБД имеют ряд средств для обеспечения поддержания целостности.

Выделяют три группы правил целостности:

1) по сущностям;

2) по ссылкам;

3) целостность доменов или целостность, определяемая пользователем**.**

Существует два правила целостности, общих для любых реляционных баз данных:

1) не допускается, чтобы атрибут, находящийся в первичном ключе, принимал неопределенное значение;

2) значение внешнего ключа должно либо быть равным значению первичного ключа цели, либо быть полностью неопределенным, т.е. каждое значение атрибута, участвующего во внешнем ключе, должно быть неопределенным.

Для любой конкретной базы данных существует ряд дополнительных специфических правил, которые относятся именно к данной базе данных и определяются разработчиком. Чаще всего контролируется:

- уникальность тех или иных атрибутов;
- диапазон значений;
- принадлежность набору значений (пол «М» или «Ж»).

8.2.3. Сжатие данных

В настоящее время при наличии запоминающих устройств с большим объемом памяти (до нескольких десятков гигабайт) проблема сжатия данных не потеряла своей актуальности. Действительно, с приходом новых технологий появилась возможность создания БД с очень большим объемом хранимой в них информации (например, распределенные БД с таблицами, содержащими гигабайты данных). Для хранения таких БД по-прежнему приходится применять технологию сжатия данных.

Естественно, что механизм сжатия данных должен быть обратим. Преимущества СУБД, используемых сжатие данных [6]:

1) в территориально удаленных СУБД передача данных от одного узла к другому требует меньше времени и, следовательно, обеспечивает более высокую скорость передачи данных по сравнению с несжатыми данными. При неавтоматической репликации данных (рабо-

ты с копией БД или объектами, допускающими синхронизацию данных) возможно использование обычных файловых архиваторов;

2) для хранения сжатых данных при резервном копировании требуется меньше устройств резервного копирования;

3) при использовании сжатия данных появляется возможность упаковывать больше ключей в блок индекса заданного размера. Используемые значения ключей сначала сжимаются, а уже потом начинают сравниваться со сжатыми ключами в самом индексе. Следовательно, если мы имеем больше ключей, хранимых в индексном блоке заданного размера, то в результате потребуется меньше операций для поиска того блока индекса, который необходим для доступа к нужным данным.

В различных СУБД могут существовать свои алгоритмы сжатия данных, однако не существует обобщающего алгоритма для обеспечения наилучшего эффекта сжатия данных. Так, например, в СУБД MS Access при сжатии базы данных индексы оптимизируются по быстродействию, т.е. для поддержания оптимизации по быстродействию необходимо регулярно выполнять сжатие базы данных. Для такой цели в этой СУБД существует специальная подпрограмма сжатия данных.

8.2.4. БД, поддерживаемые в оперативной памяти

В настоящее время существуют СУБД, способные обрабатывать данные в оперативной памяти на качественно более высоком уровне. Использование СУБД такого класса позволяет пользователям обрабатывать данные в несколько раз быстрее, чем в случае с работой при обращении непосредственно к жестким дискам. Обычно для БД, поддерживаемым в оперативной памяти, их состояние сохраняется в некоторых контрольных точках в виде дисковых копий. Такие контрольные точки возникают в периоды наименьшей активности пользователей.

8.3. Экстенсиональная и интенсиональная части базы данных

Абстрагируясь от конкретных СУБД и внимательно присмотревшись к тому, что реально хранится в базе данных, можно заметить наличие трех различных видов информации. Во-первых, это информация, характеризующая структуры пользовательских данных (описание структурной части схемы базы данных). Такая информация в случае реляционной базы данных сохраняется в системных отношениях-каталогах и содержит главным образом имена базовых отношений и имена и типы данных их атрибутов. Во-вторых, это собственно наборы

кортежей пользовательских данных, сохраняемых в определенных пользователями отношениях. Наконец, в-третьих, это правила, определяющие ограничения целостности базы данных, триггеры базы данных и представляемые (виртуальные) отношения. В реляционных системах правила опять же сохраняются в системных таблицах-каталогах, хотя плоские таблицы далеко не идеально подходят для этой цели [1].

Информация первого и второго вида в совокупности явно описывает объекты (сущности) реального мира, моделируемые в базе данных. Другими словами, это явные факты, предоставленные пользователями для хранения в БД. Эту часть базы данных принято называть экстенсиональной.

Информация третьего вида служит для руководства СУБД при выполнении различного рода операций, задаваемых пользователями. Ограничения целостности могут блокировать выполнение операций обновления базы данных, триггеры вызывают автоматическое выполнение специфицированных действий при возникновении специфицированных условий, определения представлений вызывают явную или косвенную материализацию представляемых таблиц при их использовании. Эту часть базы данных принято называть интенсиональной; она содержит не непосредственные факты, а информацию, характеризующую семантику предметной области.

Несложно заметить, что в реляционных базах данных на первом месте стоит экстенсиональная часть, а интенсиональная часть несет вспомогательную нагрузку.

Контрольные вопросы

1. Перечислите основные составляющие базы данных.
2. Назовите основные типы индексов.
3. Поясните метод доступа к данным посредством хеширования.
4. В чем заключается оптимизация работы с базой данных?

9. СУБД ТРЕТЬЕГО ПОКОЛЕНИЯ И ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ (ООСУБД)

9.1. Манифесты СУБД третьего поколения и ООСУБД

Реляционные СУБД давно подвергаются критике за то, что могут работать с весьма ограниченными по семантике наборами данных. Несовершенство реляционных СУБД послужило толчком к развитию объектно-ориентированных и объектно-реляционных СУБД.

В июле 1990 г. Комитет по развитию функциональных возможностей СУБД опубликовал доклад, представленный на конференции в Виндермере (Англия) и получивший название «Манифест СУБД третьего поколения». Полный текст документа изложен в [19], мы рассмотрим общие положения Манифеста. Авторы документа отмечают, что на момент опубликования в докладе сформировались два поколения таких систем:

первое поколение — иерархические и сетевые системы, которые были первыми системами, позволившими объединить средства языков определения данных и манипулирования данными для совокупностей записей;

второе поколение — реляционные СУБД, важный шаг в эволюции, с которым связаны использование непроцедурного языка манипулирования данными и обеспечение в существенной степени независимости данных.

Авторы манифеста сформулировали в документе три принципа и тринадцать предложений, имеющих отношение к СУБД 3-го поколения.

Принципы:

1) СУБД 3-го поколения будут обеспечивать сервисы в трех областях:

- управления данными;
- управления объектами;
- управления знаниями;

2) СУБД 3-го поколения должны относится к категории СУБД 2-го поколения в том смысле, что не должны быть утрачены непроцедурный доступ и независимость данных;

3) СУБД 3-го поколения должны быть открытыми для других подсистем. Программные продукты, претендующие на статус СУБД третьего поколения, должны располагать языками четвертого поколения (4GL), инструментарием поддержки принятия решений, средства-

ми для выполнения удаленных операций над данными, а также распределенными возможностями.

Предложения

Все предложения были условно разделены на 3 группы.

Первая группа содержит предложения, связанные с ОО-свойствами, которые необходимы для систем третьего поколения:

1) СУБД 3-го поколения должны содержать объектно-ориентированные возможности, но не как элементы полностью нового архитектурного рассмотрения СУБД, а как расширение существующих моделей;

2) СУБД 3-го поколения должны поддерживать механизм наследования;

3) СУБД 3-го поколения должны поддерживать функции (методы, процедуры) и инкапсуляцию;

4) СУБД 3-го поколения должны факультативно назначать уникальные идентификаторы для записей;

5) Правила (триггеры) должны стать важной возможностью будущих систем, но их не следует ассоциировать с какими-либо определенными функциями или коллекциями.

Вторая группа содержит предложения, касающиеся усиления функций СУБД:

1) навигация для доступа к требуемым данным должна использоваться только как крайнее средство. Другими словами, не следует возвращаться к технике доступа к данным посредством написания внешних программных конструкций, как это было в иерархических и сетевых СУБД;

2) должно быть не менее двух вариантов реализации коллекций, один из которых использует простое перечисление членов коллекции, а другой — это язык запросов для спецификации критерия членства;

3) должна существовать возможность обновления представлений;

4) кластеризация, индексы уникальных идентификаторов, буферы в пользовательском пространстве и другие подобные им аспекты являются физическими, а не логическими, и они не имеют ничего общего с моделью данных.

Третья группа предложений относится к идеологии открытых систем:

1) СУБД 3-го поколения должны быть многоязычными;

2) необходимо близкое соответствие между системами типов СУБД 3-го поколения и языков, а также должны быть исключены несоответствия различных стандартов описания языка SQL;

3) несмотря на многочисленные недостатки SQL он должен оставаться и будет играть свою роль в системах 3-го поколения;

4) SQL-запросы и ответы на них должны быть самыми нижними уровнями коммуникаций между клиентом и сервером.

После того как был опубликован Манифест СУБД 3-го поколения, эти принципы и предложения стали играть важную роль в мире реляционных БД, расширенных объектно-ориентированными возможностями.

За год до опубликования Манифеста СУБД 3-го поколения другая группа ученых в области БД опубликовала документ под названием Манифест объектно-ориентированных систем баз данных. И если в Манифесте третьего поколения сформулированы перспективные направления в разработке так называемой гибридной модели (т.е. смеси реляционной и объектно-ориентированной), то в Манифесте ОО-сис-тем сделано то же самое для сугубо объектно-ориентированных БД (ООБД). Этот документ содержит также тринадцать «заповедей» относительно обязательных свойств ООБД [20].

1. Поддержка сложных объектов.

Сложные объекты следует поддерживать. Сложные объекты строятся из более простых при помощи конструкторов. Простейшими объектами являются такие объекты, как целые числа, символы, символьные строки произвольной длины, булевские переменные и числа с плавающей точкой (можно было бы добавить другие атомарные типы). Имеются различные конструкторы сложных объектов: примерами могут служить конструкторы кортежей, множеств, мультимножеств, списков, массивов.

2. Идентифицируемость объектов.

Идея состоит в следующем: в модели с идентифицируемостью объектов объект существует независимо от его значения. Таким образом, имеется два понятия эквивалентности объектов: два объекта могут быть идентичны (они представляют собой один и тот же объект) или они могут быть равны (они имеют одно и то же значение). Это влечет два следствия: первое — разделяемые объекты, а второе — изменения объектов.

3. Объекты следует инкапсулировать.

Идея инкапсуляции происходит из потребности отчетливо различать спецификации и реализации операций и из потребности в модульности. Интерпретация этого принципа для баз данных состоит в том, что объект инкапсулирует и программу, и данные. Таким образом, инкапсуляция обеспечивает что-то вроде «логической независимости данных»: можно изменить реализацию типа, не меняя каких-либо программ, использующих этот тип.

4. Типы и классы.

Система должна предоставлять некоторый механизм структурирования данных, будь это классы или типы. Таким образом, классическое понятие схемы базы данных будет заменено на понятие множества классов или множества типов.

5. Иерархии классов или типов.

Для классов или типов следует поддерживать наследование. Наследование обладает двумя положительными достоинствами. Во-первых, оно является мощным средством моделирования, поскольку обеспечивает возможность краткого и точного описания мира. Во-вторых, эта возможность помогает факторизовать совместно используемые в приложениях спецификации и реализации.

6. Перекрытие, перегрузка и позднее связывание.

Не следует производить преждевременное связывание. Чтобы обеспечить перекрытие и перегрузку, система не может связывать имена операций с программами во время компиляции. Поэтому имена операций должны разрешаться (транслироваться в адреса программ) во время выполнения. Эта отложенная трансляция и называется ***поздним связыванием***.

7. Вычислительная полнота.

Вычислительную полноту следует поддерживать. С точки зрения языка программирования это свойство является очевидным: оно просто означает, что любую вычисляемую функцию можно выразить с помощью языка манипулирования данными системы баз данных. С точки зрения базы данных это является новшеством, так как SQL, например, не является полным языком. Однако вычислительная полнота — это не то же самое, что «ресурсная полнота», т.е. возможность доступа ко всем ресурсам системы (например, к экрану и удаленным ресурсам) с использованием внутренних средств языка. Поэтому система, даже будучи вычислительно полной, может быть не способна выразить полное приложение. Тем не менее такая система является более мощной, чем система баз данных, которая только хранит и извлекает данные и выполняет простые вычисления с атомарными значениями.

8. Расширяемость.

Система базы данных поставляется с набором предопределенных типов. Эти типы могут при желании использоваться программистами для написания приложений. Набор предопределенных типов должен быть расширяемым в следующем смысле: должны иметься средства для определения новых типов и не должно быть различий в использовании системных и определенных пользователем типов. Конечно, способы поддержания системой системных и пользовательских типов могут значительно различаться, но эти различия должны быть невиди-

мыми для приложения и прикладного программиста. Напомним, что определение типов включает определение операций этих типов. Заметим, что требование инкапсуляции подразумевает наличие механизма для определения новых типов. Требование расширяемости усиливает эту возможность, указывая, что вновь созданные типы должны иметь тот же статус, что и существующие. Однако мы не требуем, чтобы совокупность конструкторов типов (кортежей, множеств, списков и т. д.) была расширяемой.

9. Стабильность.

Данные следует помнить. Это требование очевидно с точки зрения баз данных, но является новым с точки зрения языков программирования. Стабильность означает возможность программиста обеспечить сохранность данных после завершения выполнения процесса с целью последующего использования в другом процессе. Свойство стабильности должно быть ортогональным, т.е. для каждого объекта вне зависимости от его типа должна иметься возможность сделать его стабильным (т.е. без какой-либо явной переделки объекта). Стабильность должна быть неявной: пользователь не должен явно перемещать или копировать данные, чтобы сделать их стабильными.

10. Управление вторичной памятью.

Следует уметь управлять большими базами данных. Управление вторичной памятью является классической чертой систем управления базами данных. Эта возможность обычно поддерживается с помощью набора механизмов; среди них — управление индексами, кластеризация данных, буферизация данных, выбор пути доступа и оптимизация запросов.

11. Параллелизм.

Следует поддерживать параллельную работу нескольких пользователей. Что касается управления параллельным взаимодействием с системой нескольких пользователей, то должны обеспечиваться услуги того же уровня, что и предоставляемые современными системами баз данных. Поэтому система должна обеспечивать гармоническое сосуществование пользователей, одновременно работающих с базой данных. Следовательно, система должна поддерживать стандартное понятие атомарности последовательности операций и управляемого совместного доступа. Должна, по крайней мере, обеспечиваться сериализация операций, хотя могут существовать и менее жесткие альтернативы.

12. Восстановление.

Следует обеспечивать восстановление после аппаратных и программных сбоев. И в этом случае система, должна обеспечивать услуги того же уровня, который предоставляют современные системы баз данных.

13. Средства обеспечения незапланированных запросов.

Средство обеспечения запросов должно удовлетворять следующим трем критериям:

– быть средством высокого уровня, т.е. пользователь должен иметь возможность кратко выразить нетривиальные запросы (в нескольких словах или несколькими нажатиями клавиш мыши). Это означает, что средство формулирования должно быть достаточно декларативным, т.е. упор должен быть сделан на «что», а не на «как».

– быть эффективным, т.е. формулировка запросов должна допускать возможность оптимизации запросов.

– Средство запросов не должно зависеть от приложения, т.е., оно должно работать с любой возможной базой данных. Это последнее требование устраняет потребность в специфических средствах обеспечения запросов для конкретных приложений и необходимость написания дополнительных операций для каждого определенного пользователем типа.

Помимо обязательных «заповедей», в Манифесте ООСУБД содержались также дополнительные положения, характерные для ООСУБД:

– множественное наследование;
– проверка и вывод типов;
– распределенность;
– проектные транзакции;
– версии;
– парадигма программирования;
– система представления;
– система типов;
– однородность.

Имеется ряд коммерческих объектно-ориентированных систем баз данных. В их числе GemStone (от Servio Corporation), ONTOS (ONTOS), Object Store (Object Design, Inc.), Objectivity/DB (Objectivity,Inc.), Versant (Versant Object Technology, Inc.), Object Database (Object Database, Inc.), Itasca (Itasca Systems, Inc.), 02 (02 Technology). Все эти продукты поддерживают объектно-ориентированную модель данных. Они позволяют пользователю создавать новый класс с атрибутами и методами, определять наследование атрибутов и методов у суперклассов, создавать экземпляры поодиночке или группами, загружать и выполнять методы.

Чем же различаются эти два похода к новым типам СУБД?

Оба документа предполагают поддержку сложных объектов, инкапсуляции, структуры классов, наследования и расширяемости [19, 20]. Однако использоваться эти механизмы будут по-разному. Наибо-

лее крупные из отличий между положениями, сформулированными в двух манифестах, — это такие механизмы, как уникальные идентификаторы объектов, поддержка полиморфизма и языковые средства. Здесь позиции авторов полностью расходятся. Сторонниками объектно-ориентированного подхода предлагается введение уникальных идентификаторов объектов, поддержка полиморфизма и использование вычислительно полных языков для работы с базами данных. Их же идейные противники считают необходимым использование уникальных идентификаторов только в случае, если объект не содержит уникального ключа, полностью не приемлют полиморфизм и в качестве основного языка для работы с базами данных предлагают некоторое развитие языка SQL.

9.2. Общие понятия ОО-подхода к БД

Одним из общих аргументов в пользу объектно-ориентированных систем является наличие возможности более мощного и более высокоуровневого семантического моделирования по сравнению с системами, в основе которых лежат ER-модели.

Одной из основных причин, по которым предпочтительней использовать СУБД с ОО-интерфейсом заключается в том, что ОО-приложения могут взаимодействовать непосредственно с ОО-частью такой СУБД. Объектно-ориентированный подход базируется на концепциях:

– объекта и идентификатора объекта (индивидуальность объекта);
– атрибутов и методов;
– классов;
– иерархии и наследования классов.

Объектный тип — это расширение типа, определяемого пользователем, позволяющее инкапсулировать методы с элементами данных в едином логическом модуле. Определение объектного типа служит в качестве шаблона, но не распределяет память. Объекты хранятся физически как строки или столбцы таблицы.

Любая ***сущность*** реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом во все время его существования и не меняется при изменении состояния объекта.

Таким образом, аналогично строке таблицы объект представляет данные, а объектный тип — их структуру.

Каждый объект имеет состояние и поведение. Состояние объекта — набор значений его атрибутов. Поведение объекта — набор ***методов*** (программный код), оперирующих над состоянием объекта. Определение метода в ООБД производится в два этапа. Сначала объявляется сигнатура метода, т.е. его имя, класс, типы или классы аргументов и тип или класс результата. Методы могут быть публичными (доступными из объектов других классов) или приватными (доступными только внутри данного класса). На втором этапе определяется реализация класса на одном из языков программирования ООБД, например в СУБД Oracle это подпрограммы на языке PL/SQL или на C, которые определяют набор допустимых операций для конкретного объектного типа. Иногда методы называют поведенческой частью объектного типа.

Значение атрибута объекта — это тоже некоторый объект или множество объектов. Состояние и поведение объекта инкапсулированы в объекте; взаимодействие между объектами производится на основе передачи сообщений и выполнении соответствующих методов.

Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Объект должен принадлежать только одному классу (если не учитывать возможности наследования). Допускается наличие примитивных предопределенных классов, объекты — экземпляры которых не имеют атрибутов: целые, строки и т.д. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется доменом этого атрибута.

Допускается порождение нового класса на основе уже существующего класса — наследование. В этом случае новый класс, называемый подклассом существующего класса (суперкласса), наследует все атрибуты и методы суперкласса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различаются случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного суперкласса, во втором случае суперклассов может быть несколько. При поддержании множественного наследования классы связаны в ориентированный граф с корнем, называемый решеткой классов. Объект подкласса считается принадлежащим любому суперклассу этого класса.

Одной из более поздних идей объектно-ориентированного подхода является идея возможного переопределения атрибутов и методов суперкласса в подклассе (перегрузки методов). Эта возможность увеличивает гибкость, но порождает дополнительную проблему: при компиляции объектно-ориентированной программы могут быть неизвестны структура и программный код методов объекта, хотя его класс (в общем случае — суперкласс) известен [1]. Для разрешения этой про-

блемы применяется так называемый метод позднего связывания, означающий, по сути дела, интерпретационный режим выполнения программы с распознаванием деталей реализации объекта во время выполнения посылки сообщения к нему.

Специфической особенностью модели ООБД является возможность объявления дополнительных «исключительных» атрибутов и методов для именованных объектов. Это означает, что конкретный именованный объект-представитель класса может обладать типом, являющимся подтипом типа класса [1].

Объектная база данных обеспечивает доступ к различным источникам данных. В том числе, конечно, и к данным реляционных СУБД, а также разнообразные средства манипуляции с объектами базы данных.

9.3. Реализация ОО-подхода в СУБД Oracle

СУБД Oracle, начиная с версии 8/8i предоставляет расширенный набор встроенных типов данных, а также позволяет конструировать новые типы данных со спецификацией методов доступа к ним. Это означает фактически наличие инструмента, позволяющего строить структурированные типы данных, непосредственно отображающие сущности предметной области. В данном разделе будем опираться на материалы изложенными в [18, 21].

Oracle 8 уже фактически опирался на новый стандарт SQL, позволяющий описывать определения новых типов объектов, состоящих из атрибутов (скалярных — т.е. других типов, множеств объектов, ссылок на объекты), и обладающих ассоциированными с ним методами. Любая колонка таблицы может быть любого типа, поддерживаются также вложенные таблицы и массивы объектов переменной длины.

Среди ОО-возможностей СУБД ORACLE версий 8/8i, 9/9i, 11/11i можно выделить следующие компоненты и функциональные реализации**:**

– объектные типы (записи или классы);

– объектные представления, которые позволяют собрать воедино несколько нормализованных отношений в одном бизнес-компоненте;

– объектный язык. Расширения языков Oracle SQL и PL/SQL;

– объектный API-интерфейс. Объекты поддерживаются с помощью предкомпилятора Oracle (например, Pro*C), PL/SQL или OCI;

– переносимость объектов. Например, благодаря транслятору объектных типов Object Type Translator (OTT) можно преобразовывать объектные типы Olacle8 в классы C++.

В последних версиях Oracle определяемые пользователем типы данных могут применяться в качестве:

- столбца реляционной таблицы;
- атрибута внутри другого объектного типа;
- части объектного представления реляционных таблиц;
- основы объектной таблицы;
- основы переменных языка PL/SQL.

В состав расширенного языка Oracle SQL входят перечисленные ниже операторы, предназначенные для управления объектными типами:

CREATE TYPE;
ALTER TYPE;
DROP TYPE;
GRANT TYPE;
REVOKE TYPE.

Подобно столбцам отношения, атрибуты внутри объектного типа должны быть уникальными, но они могут повторно использоваться в других объектных типах. Так же, как и при работе со столбцами, доступ к атрибутам осуществляется на основе их имен. При этом не предусмотрены никакие формы доступа на основе известной позиции или смещения. Любой объектный тип может быть простым, составным или самоссылочным. Простой объектный тип соответствует своему названию, а составной отличается от него тем, что может содержать, по крайней мере, еще один объектный тип. Самоссылочный объектный тип содержит минимум один объект, который ссылается на этот же объектный тип. Ниже представлены примеры использования объектных типов.

Пример 9.1. Создание объектного типа сотрудника.

```
CREATE type employee_type as object (
ssn umber(9),
name varchar2(35),
address varcher2(70),
resume varchar2 (250));
```

Пример 9.2. Использование объектного типа в качестве столбца реляционной таблицы.

```
CREATE table manager_candidates (
 position varchar2(40),
 vacancy varchar2(250),
 employee employee_type);
```

```
INSERT INTO manager_candidates (
 ‘Технический менеджер ’, ‘свободная вакансия’,
 emploee_type(12345, ‘Иванов И.И.’,
‘г. Томск, ул. Ленина 21’, ‘Научный работник’));

SELECT mc.employee.name
FROM manager_candidates mc
WHERE mc.employee.ssn=12345;
```

Пример 9.3. Создание таблицы на основе объектного типа.

```
CREATE table employee of employee_type;
```

Следует отметить, что для извлечения значения атрибута объекта внутри таблицы необходимо использовать псевдоним и точечную нотацию (пример 9.2).

При создании объектной таблицы создается столбец с идентификатором объекта (OID или REF) для экземпляра объекта (строки), что отличается от создания реляционной таблицы с объектным типом в качестве столбца, поскольку в таком случае объектный тип не имеет OID-идентификтора.

Объектный тип может иметь методы MEMBER — это подпрограмма, которая оперирует данными, т.е. атрибутами внутри любого объектного типа. Метод является процедурой или функцией языка PL/SQL.

Для каждого метода необходимо создать интерфейс (или спецификацию) и внутреннюю реализацию (тело метода). Для доступа к методу используется нотация с точкой типа object.method. В следующем примере создается метод некоторого объектного типа со спецификацией и телом, а также фрагмент кода PL/SQL с вызовом этого метода.

Пример 9.4. Создание и использование метода.

```
CREATE type auto_type as object (
Vin varchar2(80),
Make varchar2(30),
Model varchar2(40),
member function getcost
return number,
pragma restrict_references
(getcost, WPNS));
/* статус WPNS – Writes No Package State, означает, что функция не может изменять никакие внутренние параметры. */
create type body auto_type (
member function getcost
return number is
```

```
begin
return (cost); /*виртуальная функция вычисления стоимости автомобиля*/
end;);
/*необходимо создать таблицу auto */
declare
    a auto_type;
    c number;
    begin
            select value (a)
            into a
            from auto a
            where a.vin=122;
            c:=a.getcost();
end;
```

Коллекции (массивы VARRAY и вложенные таблицы)

Коллекцией называется упорядоченная или неупорядоченная группа элементов. В ORACLE 8 существует два таких типа: массив VARRAY является упорядоченной коллекцией, а вложенная таблица TABLE — неупорядоченной. Вложенная таблица TABLE входит в стандарт ANSI. Пояснить различия VARRAY и TABLE можно на простом примере отношения «один-ко-многим» типа основные сведения – подчиненные сведения. Для списка размещенных по порядку предметов используется коллекция VARRAY, а для группы подчиненных сведений для каждого основного предмета можно использовать вложенную коллекцию TABLE внутри коллекции VARRAY. Вложенная таблица является определяемым пользователем типом, или типом TABLE, который может применяться внутри таблицы как столбец, а внутри объектного типа — как атрибут или переменная PL/SQL.

Вложенные таблицы сохраняются вне основной таблицы в так называемой таблице хранения (s*torage table*). Массив VARRAY содержит счетчик (count), определяющий количество элементов, которые находятся в настоящее время в массиве и предел (limit) – максимальное количество элементов, которое может содержать массив. Коллекция VARRAY аналогична структуре данных типа массив, а TABLE — таблице.

Поддержка OLAP

Разработчики Oracle, понимая, что реляционная модель удобна для представления данных в информационно-управляющих системах, убеж-

дены, что для аналитических систем более подходит многомерная модель, где данные представлены в виде многомерных кубов, которые можно легко вращать, получать срезы, агрегировать информацию и т. д. Для создания OLAP-приложений в Oracle ранее использовался программный продукт Express Server — СУБД с многомерной моделью. Данные из оперативных реляционных систем приходилось перегружать или подкачивать в Express Server, который не обеспечивал такого же уровня надежности, масштабирования, защиты, как реляционный сервер Oracle. Поэтому в версию 9.2 интегрирована возможность и функциональность Express Server. Теперь сервер Oracle 9i поддерживает и многомерную модель данных, что позволяет пользователю проектировать многомерные кубы и решать, как они будут храниться в Oracle 9i — в реляционных таблицах или в так называемых аналитических пространствах (LOB-поля). Обеспечивается возможность переноса данных из базы Express Server в Oracle 9i. Реализован весь набор функций, ранее присущий Express, причем разработчикам Oracle удалось добиться того, что скорость выполнения этих функций была не ниже, чем в Express Server. Кроме того, метаданные и данные хранятся теперь в единой базе данных Oracle, а для работы с многомерными кубами используются JavaBeans, входящие в состав инструментария.

Поддержка XML в Oracle 9x

Сервер Oracle теперь поддерживает не только реляционную, объектную, многомерную модель данных, но и XML. В версии 9.1 большие объемы XML-данных можно было поместить в виде XML-файлов в LOB-полях как единый кусок текста. Иначе содержимое XML-файла разбиралось и «разбрасывалось» по реляционным таблицам, а при запросе вновь собиралось в файл. В версии 9.2 поддерживаются XML-схемы и можно хранить XML-данные еще и в виде собственно XML-объектов: таблиц с типом XMLType и колонок типа XMLType. В новой версии реляционные и XML-данные сосуществуют в одной универсальной модели. С XML-данными можно работать посредством языков SQL и Java, а с реляционными — через XML-интерфейсы, например, через XPath. Поскольку из SQL можно работать с XML-данными и их частями, то теперь легко построить, например, обычный индекс по реквизиту, содержащемуся в XML-файлах и быстро находить нужные файлы. Можно построить реляционное представление (View), колонками которого будут реквизиты XML-файлов и далее работать с этим представлением обычными «реляционными» средствами. Предположим, что в некотором приложении запрос на товары приходит от заказчика в виде сообщений, содержащих XML-текст. Можно написать запрос, одновре-

менно работающий с реляционными данными, очередями сообщений, XML-данными, пространственными данными, контекстом (рис. 1).И наоборот, создав над реляционными или объектными таблицами базы данных представление XMLType View, можно работать с этими данными через XML-интерфейс. В Oracle 9.2 поддерживаются стандарты доступа SQLX, Xpath, DOM, JavaBeans и JNDI.

Oracle9i обеспечивает возможность оперировать с XML-документами так же, как и с реляционными данными. Подсистема, названная Oracle XML DB, вошла в состав второго релиза Oracle9i.

Oracle XML DB дает пользователям возможность, наряду с реляционными данными, применять XML-документы, не преобразуя их в строки и колонки. XML-документы храняться как объекты Oracle9i, что сделано для совместимости с Oracle8.

Разработчики заявляет, что Oracle XML DB — единственное средство для работы с XML-документами и реляционными данными в рамках одной базы данных, причем они доступны в едином запросе.

Объектные представления

Объектные представления реляционных таблиц в Oracle позволяют разработчикам и пользователям применять такие преимущества, как соединение объектно-ориентированных пользовательских приложений и реляционных хранилищ данных, согласование нескольких объектно-ориентированных пользовательских схем с одной реляционной основной моделью, а также параллельную разработку новых объектно-ориентированных и текущих реляционных приложений. Последнее означает, что объектные представления упрощают процесс миграции к объектно-ориентированным технологиям. При этом пользователь может работать со всей системой так, как если бы она была объектно-ориентированной, хотя на самом деле данные являются реляционными.

Общий порядок реализации объектного представления состоит из создания следующих объектов:

- таблиц;
- объектных типов;
- объектных представлений;
- триггеров INSTEAD OF.

Назначение триггера INSTEAD OF позволяет вставлять, обновлять и удалять данные реляционных таблиц, на которых основано объектное представление, вместо непосредственной модификации объектного представления, что невозможно.

Пример 9.5. Этапы создания объектного представления.

```
CREATE table depts(
    deptid number,
    deptbudget number);
CREATE table branches(
branchid number,
branchbudget number,
deptid number);
CREATE type funding_type as object(
    deptid number,
    deptbudget number,
    branchid number,
branchbudget number);
CREATE OR REPLACE view funding_objv as
    SELECT funding_type
    (deptid, deptbudget, branchid, branchbudget) funding
    FROM depts d, branches  b
    WHERE d.deptid=b.deptid;
CREATE OR REPLACE trigger funding_trig INSTEAD OF
    INSERT ON funding_objv for each row
    Begin
    Insert into depts values
(:new.funding.deptid, :new.funding.deptbudget);
Insert into branches values
(:new.funding.branchid, :new.funding.branchbudget,
:new.funding.deptid);
    end;
```

Объектные представления похожи на традиционные реляционные представления, но в них могут использоваться строгая типизация, сложные структуры (не в первой нормальной форме), методы и возможности ссылок на существующие реляционные данные.

В заключение данного раздела хотелось бы отметить, что объектно-ориентированные СУБД, имеющие связь с классическими реляционными моделями данных имеют хорошие перспективы для развития, обусловленные совершенствованием технологий связи, быстрым развитием Internet, постоянным ростом используемых объемов данных. Возможность гибкой настройки схемы данных, а также способность хранить большое количество объектов делают оправданным применение ООСУБД и гибридных СУБД на крупных предприятиях.

Следует также отметить, что с каждой новой версией разработчики Oracle вносят все новые и новые объектно-ориентированные возможности и усовершенствуют уже существующие.

Контрольные вопросы

1. Опишите основные различия Манифестов ООБД и СУБД 3-го поколения.
2. Охарактеризуйте общие понятия ОО-подхода к БД.
3. Перечислите основные ОО-возможности СУБД ORACLE.

10. СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

10.1. Системы управления базами данных первого поколения

10.1.1. Общие характеристики СУБД 1-го поколения

Как было отмечено в предыдущей главе, можно выделить три поколения СУБД. В последней части нашего курса мы проведем обзор СУБД всех категорий, начиная с иерархических и сетевых, заканчивая реляционными и СУБД 3-го поколения.

Широкое распространение первые СУБД получили в середине 70-х, начале 80-х годов, хотя первые наработки в области создания баз данных появились в конце 60-х. В основу построения некоторых наиболее ранних СУБД положены иерархическая и сетевая модель данных. Нельзя не отметить переходный тип СУБД, приближенных к реляционным, — системы с инвертированными файлами.

При создании первых СУБД на первый план выступала способность систем хранить данные сложной структуры и значительного объема и использовать установленные связи между информационными элементами при проектировании приложений. Другое важное достоинство — это обеспечение относительной независимости программ от структур хранения в том смысле, что если в БД происходили структурные изменения, не затрагивающие подсхему программы (части структуры БД, доступной программе), то не было необходимости вносить изменения в программу. Внешнее взаимодействие с такими СУБД осуществлялось путем обращения к программе СУБД из программы приложения, написанной на одном из базовых языков программирования (в то время это были Ассемблер, КОБОЛ. PL/1), и такие системы стали называть системами с базовым языком. При этом СУБД выполняла лишь простые операции выборки записей, удовлетворяющих определенным условиям и в определенной последовательности (навигация по структуре), а также операции включения, замены и удаления записей. Но все эти операции осуществлялись с учетом зафиксированной структуры БД, что существенно сокращало алгоритмическую часть программы, касающуюся согласованной выборки связанных записей, и снижало риск нарушения структурной целостности БД [2].

СУБД первого поколения были системы IMS (иерархическая СУБД), IDS (сетевая СУБД), ADABAS (СУБД с инвертированными файлами) и соответствующие им отечественные СУБД ОКА, БАНК-ОС, ДИСОД.

10.1.2. СУБД IMS (ОКА)

IMS (Information Management System) фирмы IBM и ОКА фирмы НИЦЭВТ (г. Москва) являлись весьма распространенными СУБД, обеспечивающими хранение и доступ к БД иерархической структуры [2].

Физическая база данных представляется в виде древовидной структуры. Элементом структуры является сегмент, который может состоять из одного или нескольких полей (данные – символьные или числовые (десятичный и двоичный форматы)). Для СУБД IMS характерны следующие свойства:

– в БД может быть только один тип корневого сегмента;

– у корневого сегмента могут быть порожденные сегменты;

– каждый порожденный сегмент может также иметь порожденные сегменты (не более 15 сегментов в любом из путей);

– одна база данных может содержать до 255 типов сегментов;

– для каждого экземпляра определенного типа сегмента может существовать произвольное число экземпляров каждого из порожденных им сегментов;

– порожденный сегмент существует только при наличии исходного.

Сегменты одного типа имеют единую для этого типа внутреннюю структуру и размер, в том числе фиксированные размеры и типы одноименных полей в различных реализациях сегментов. Иерархия типов сегментов определена сверху вниз и слева направо.

Подчиненные типы сегментов не содержат ключей старшего сегмента, поэтому подчиненные не могут существовать без «своих» старших. Однако при выборке сегментов некорневого уровня ключи старших (до корневого) присоединяются к выбираемой реализации сегмента, обеспечивая тем самым его семантическую целостность.

Каждому экземпляру корневого сегмента соответствует физическая запись. Физические базы данных могут расщепляться на несколько файлов, но файл не может содержать сегменты принадлежащих различным физическим базам данных. Пример записи базы данных ВУЗ представлен на рис. 10.1.

IMS (ОКА) поддерживает 4 способа физической организации хранения и доступа к данным [7]:

- иерархический последовательный метод доступа;
- иерархический индексно-последовательный метод доступа;
- иерархический индексно-прямой метод доступа;
- иерархический прямой.

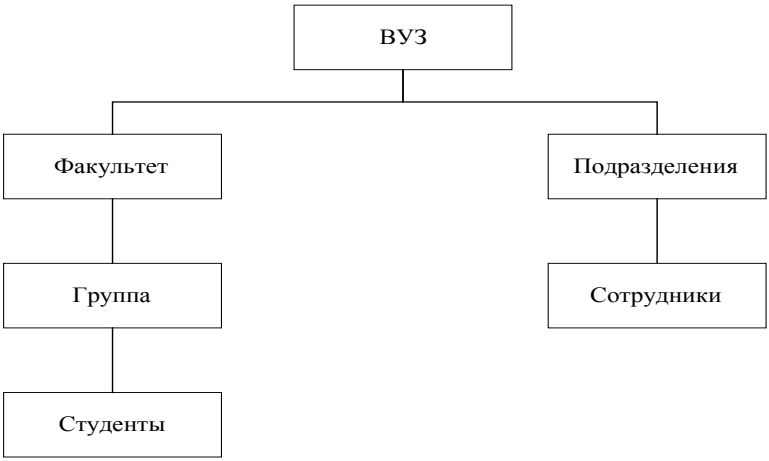

Рис. 10.1 Пример иерархической структуры БД

Информационный обмен прикладной программы с БД осуществляется с помощью оператора **CALL**, вызывающую программу-СУБД с указанием в качестве параметров: функции вызова; подсхемы, связывающие программу с доступной ей частью структуры данных; области ввода-вывода; аргумента поиска, определяющего имена сегментов иерархического пути, участвующих в обмене, и условия отбора реализаций сегментов [2].

Система IMS (ОКА) обеспечивает выполнение всех типичных для СУБД функций: вставку, замену, удаление и чтение реализаций сегментов [2]. Вставка реализаций сегментов может осуществляться в соответствии со значением ключевого поля (по возрастанию значений) или в соответствии с одним из следующих правил: первая среди подобных; последняя среди подобных или ближайшая к текущей позиции БД. Замена или удаление реализаций сегментов выполняется только после успешного завершения процедур чтения заменяемых или удаленных реализаций сегментов. Допускается одновременная замена сегментов одного пути, но запрещается замена ключевых полей. При удалении неконцевых сегментов удаляются и все реализации подчиненных ему сегментов.

Характерной особенностью IMS (ОКА) является значительное число модификаций основной функции — чтения реализаций сегментов. Система обеспечивает возможности поиска требуемых сегментов с начала БД или от текущего состояния; поиска только среди подчиненных специально определенной реализации старшего сегмента; по-

следовательного просмотра сегментов независимо от их типа (в иерархическом порядке); одновременной выборки нескольких сегментов иерархического пути; поиска реализаций сегментов, удовлетворяющих комбинациям ограничений на значение полей, в том числе полей вышестоящих сегментов и др. [2].

Важными особенностями системы являются возможность запоминания и дальнейшего использования позиции в БД (места последнего обращения), применение кодов команд, модифицирующих функцию вызова, а также динамического изменения всех параметров вызова. Эти возможности позволяют в большинстве случаев упростить алгоритмы прикладных программ [2].

10.1.3. СУБД IDS (БАНК-ОС)

На формирование концепций сетевых СУБД огромное влияние оказала ассоциация КОДАСИЛ (Conference On Date System Languages - CODASYL), образованная в 1959 году, состоящая из представителей 40 организаций. В круг задач этой ассоциации входила разработка методов и языков, используемых при работе с БД. Первым проектом, реализованным ассоциацией КОДАСИЛ, явился язык КОБОЛ. Позднее из ассоциации выделилась группа, проводящая работы по расширению возможностей языка КОБОЛ для обработки баз данных, названная РГБД (DBWG — Рабочая группа по Базам данных). С участием этой группы было создано множество СУБД, в том числе и СУБД IDS фирмы Honeywell (БАНК-ОС НИИ УМС, г. Пермь — отечественный аналог IDS), обеспечивающая хранение и доступ к БД с сетевой структурой.

РГБД специфицировала два способа установления взаимосвязей между типами записей: с помощью цепей и с помощью массива указателей. В IDS использовался первый способ. Основной единицей, обеспечивающей сетевую структуру, является цепь. В цепь объединяются информационно зависимые логические записи. Запись представляет собой основную единицу информации и состоит из полей данных и служебных полей. Служебные поля предназначены для идентификации записи и организации цепей с помощью адресных ссылок. Информационные поля (поля данных) могут содержать числовые и символьные данные и имеют имена (до 8 символов). Записи одного типа содержат одноименные данные и имеют фиксированную длину и одноименные поля. Физически в записи хранится соответствующий имени код типа записи, занимающий 1 байт [2]. Имя записи употребляется в программах при обращении к записи. В БД может быть до 255 типов записей. Записи адресуются в БД с помощью адресных ссылок, состоящих из номера

«страницы» БД (БД предварительно разбивается на «страницы») и номера байта начала записи на «странице».

Цепи образуются с помощью адресных ссылок, хранящихся в записи и указывающих на следующую запись в этой цепи. Каждой цепи присваивается уникальное имя. В БД может быть определено до 500 типов цепей. Цепь обязательно включает в себя одну главную и некоторое количество детальных записей. Главная запись имеет адрес первой детальной записи, та в свою очередь — второй и т.д. Последняя запись цепи содержит ссылку на главную запись, замыкая тем самым цепь.

Чаще всего все детальные записи в одной цепи представлены записями одного типа, однако IDS допускает и разнотипные детальные записи в одной цепи. По существу отдельная цепь IDS соответствует иерархической связи исходный (главная запись) — порожденный (детальная запись) в системе IMS (ОКА) при условии, что детальные записи цепи — являются записями одного типа.

Каждая запись может входить в любое количество цепей, причем допускается включение одной и той же записи в одну цепь в качестве главной, а в другую — как детальной. Одна и та же запись может быть главной для нескольких цепей. Это также соответствует возможностям IMS (ОКА), однако в IDS (БАНК-ОС) запись может быть детальной в более чем одной цепи (иметь несколько главных). Допускается и несколько цепей-связей между двумя типами записей. Порядок следования детальных записей в цепи может быть установлен в соответствии с возрастанием или убыванием значения некоторого ключевого поля.

Возможна одновременная организация цепи в прямом и обратном направлении, допускается и третья ссылка от каждой детальной на свою главную, что обеспечивает оптимизацию движения по БД.

Система также обеспечивает типовые операции доступа к данным — запоминание, выборку, модификацию и удаление записей из программ, написанных на классических языках программирования Ассемблер, Кобол, PL/1. Запоминание (включение в БД) осуществляется по адресу (возможно вычисляемому) или согласно способу упорядочения записей в цепи либо в начало, либо в конец цепи, либо в текущее положение. Поиск производится по значениям ключевых полей либо по прямому адресу, либо от текущего состояния последовательно по цепи. Модификация и удаление всегда выполняются над текущей записью, которая предварительно должна быть найдена одной из процедур поиска. Удаление записи происходит сразу во всех цепях, где она является детальной, при удалении главной записи удаляются все детальные записи цепи, однако, если детальная входит в несколько цепей, то удается лишь адресная ссылка цепи [2].

10.1.4. СУБД ADABAS (ДИСОД)

Некоторые СУБД первого поколения допускали отсутствие какой-либо формальной структуры данных. В таких системах отсутствовала необходимость связывания записей в иерархию или сеть, не требовалось наличие ни исходных, ни порожденных, ни подобных типов, то есть записи могли быть просто записями, и БД такой СУБД могла состоять из нескольких независимых файлов. При этом файлы инвертировались относительно нескольких полей и могли быть связаны с помощью метода инверсии. Взаимосвязь может быть как постоянной, так и устанавливаемой по ходу выполнения запросов. Адаптируемая система баз данных — ADABAS (ДИСОД), поставляемая фирмой Software AG of North America, — одна из наиболее удачных систем, использующих инвертированные файлы.

ADABAS поддерживает сетевую модель данных, состоящую из множества файлов с двунаправленными, типа «многие ко многим», взаимосвязями между записями. Взаимосвязи устанавливаются с помощью одинаковых полей различных файлов. Кроме того, система поддерживает иерархические представления основывающихся на общих дескрипторах (ключах) различных файлов. Для любого файла может устанавливаться до 80 связей с другими файлами, но между любой парой файлов — только одна связь.

БД в ADABAS является совокупность связанных данных, скомпонованных в виде файлов. БД системы логически и физически разделяются на две основные области: накопитель (область хранения данных) и ассоциатор (системная область, используемая для ускорения поиска по значениям некоторых атрибутов) [7].

Все записи файла ADABAS определяются одинаково. В файле любое поле, подполе или сочетание полей можно задать как дескриптор (ключ) как в момент создания файла, так и при работе с уже созданным файлом БД с помощью программы обслуживания. Допустимые форматы атрибута — символьный, десятичные упакованный и распакованный, двоичный. Некоторые атрибуты определяются как поисковые и используются в дальнейшем для квалификации записей при их поиске в БД. По значениям поисковых атрибутов формируются поисковые инвертированные списки, хранимые в ассоциаторе системы, являющиеся характерной особенностью системы ADABAS (ДИСОД).

В ADABAS записи хранятся сжатыми в виде строк байтов переменной длины. Сжатие происходит на системном уровне, не доступном для пользователя, однако пользователь может отменить сжатие

того или иного файла. Оно выполняется при загрузке файла последовательно, поле за полем, по следующему алгоритму [7]:

– конечные пробелы буквенно-текстовых полей исключаются;

– лидирующие нули числовых данных исключаются;

– упаковываются десятичные числовые данные;

– ряд последовательных пустых полей представляется одним счетчиком пустого поля.

Каждый файл БД может содержать до $16{,}7 \cdot 10^6$ однотипных записей (число файлов в БД — до 255). Каждой записи в ADABAS назначается уникальный ссылочный номер ISN (internal sequence number). Во время загрузки записи файла размещаются физически последовательно. Система ведет словарь данных, где хранятся описания БД, файлов, атрибутов, связей между файлами; словари значений атрибутов; описания программ; сведения о пользователях

Физическая организация БД в ADABAS (ДИСОД) обусловливает возможность последовательного (без использования инвертированных списков) и прямого (с их использованием) доступа к записям. Результатом поиска может быть перечень ISN или значения требуемых атрибутов из записей БД.

Последовательный доступ обеспечивает выборку записей по списку ISN, в порядке возрастания ISN, в физической последовательности ISN, в логической последовательности по значениям заданного поискового атрибута.

ADABAS (ДИСОД) осуществляет поиск по запросам пользователей записей в БД, доступ к хранимым данным, обработку данных и выдачу результатов в виде справок и сводок заданной формы. Запросы вводятся в диалоговом режиме с видеотерминалов и в пакетном режиме в образе перфокарт. Языки запросов высокого уровня ориентированы на пользователей-непрограммистов. Обеспечивается и мультидоступ к БД.

ADABAS (ДИСОД) дает возможность обращаться к БД и из прикладных программ, написанных на языках АССЕМБЛЕР, КОБОЛ, ФОРТРАН и PL/1 , в том числе под управлением телемонитора.

СУБД ADABAS имеет встроенный язык и соответствующие программные средства диалоговой обработки данных, что позволяет осуществлять удобное, по сравнению с предшествующими СУБД, взаимодействие как пользователя так и разработчика с базой данных.

10.2. Системы управления базами данных второго поколения — реляционные СУБД

10.2.1. Общие сведения

В начале 80-х годов с появлением персональных ЭВМ получают широкое распространение СУБД, позволяющие оперировать данными, представленными в виде реляционной модели и использующие язык манипулирования данными SQL, QBE. К классу реляционных систем относятся следующие СУБД:

– FoxPro (а также другие реализации систем на базе СУБД dBase), разработанная фирмой Fox Software и впоследствии купленная Microsoft;

– Access — разработка фирмы Microsoft;

– Oracle — разработка фирмы Oracle;

– MS SQL — разработка фирмы Microsoft.

– SQL Base — разработка фирмы GUPTA, впоследствии перекупленная фирмой Centura.

Этот список можно продолжить, поскольку на рынке СУБД в настоящее время царит изобилие. Несмотря на это идеологии реляционных СУБД имеет много общего:

– в основе представлений данных лежат плоские таблицы, аналогичные понятию отношений реляционной модели данных;

– строки таблицы аналогичны понятию кортежа отношения;

– поля таблицы аналогичны атрибуту;

В современных реляционных СУБД широко используется понятие домена, первичного и внешнего ключа. Поле таблицы реляционной СУБД не может принимать множественное значение, т.е. в одной строке записи не может быть несколько значений одного поля.

Все реляционные СУБД можно разделить на два типа: файл-серверные и клиент-серверные СУБД. Принципиальное различие двух подходов состоит в технологии взаимодействия программ-приложений и СУБД.

Модель или архитектура клиент-сервер является формой распределенной обработки, где вычислительные мощности разделяются среди объединенных, связанных сетью компьютеров. Приложение функционально разделено на две или более программы, которые выполняются на различных компьютерах и связаны друг с другом путем передачи сообщений по сети.

Приложения-клиенты выполняются на рабочей станции пользователя, приложение-сервер выполняется на более мощном компьютере

- сервере. Клиент посылает запросы серверу, получая результаты обработки запроса.

Рационально поручать мощному серверу выполнять крупные задачи, а клиентскому компьютеру посредственные задачи, насколько это возможно в конкретной реализации. В этом заключается *совместная обработка данных*. Задачи обработки БД, выполнения вычислений и другие задачи, требующие высокой производительности, выполняются сервером, тогда как клиент занят диалогом и графическими изображениям. База данных, построенная с учетом сети и совместной обработки, называется распределенной базой данных.

При файл-серверном подходе обработка запроса к БД происходит непосредственно на компьютере, с которого этот запрос был послан средствами самого приложения.

СУБД FoxPro, MS Access являются классическими файл-серверными СУБД, однако с их помощью (в их среде) можно создавать клиент-серверные приложения, используя в качестве серверов БД какие-либо другие СУБД (Oracle, MS SQL и т.д.).

10.2.2. СУБД FoxPro

СУБД FoxPro относится к классу dBase-систем. Эволюция СУБД семейства dBase прослеживается от dBASE к dBASEII → dBASEIII (русифицированная версия РЕБУС) → FoxBase (КАРАТ) → FoxPro различных версий под MS DOS → СУБД FoxPro для Windows и заканчивается Visual FoxPro.

Вся информация СУБД хранится в файлах на жестком диске. Файл данных представляет собой таблицу, каждая строка (запись) которой содержит сведения об описываемом объекте. Все записи БД имеют идентичную, задаваемую пользователем структуру и размеры.

В FoxPro можно обрабатывать несколько типов файлов, для которых установлены стандартные расширения [22]:

DBF — файл базы данных, к ним в FoxPro относится термин – База Данных;

fpt — файл примечаний, в котором хранятся мемо-поля БД;

idx — индексный файл;

cdx — мультииндексный файл;

prg — программный файл;

fxp — откомпилированный командный файл prg;

mem — файл для сохранения временных переменных.

DBF-файлы в FoxPro являются основными носителями данных и могут содержать до 1 млрд. записей. Размер записи до 4000 байт. Чис-

ло полей до 255. Одновременно может быть открыто до 25 БД. Файл БД может содержать поля следующих типов данных: символьных, числовых, логических и типа даты. Мемо-поля хранятся отдельно от основного файла БД в файле примечаний, связанном с основным файлом по специальной ссылке: в каждой записи DBF-файла имеется фиксированная ссылка на каждое имеющееся в БД мемо-поле. FPT-файлы являются подчиненными по отношению к DBF-файлам. В FoxPro имеются специальные команды предназначенные для работы с мемополями.

Один DBF-файл может иметь любое число индексов, и все они могут быть одновременно открыты с помощью команды Set Index или Use. При вводе, удалении или изменении записей все индексные файлы будут соответствующим образом изменяться. Главным управляющим индексом, т.е. индексом, в соответствии с которым будет перемещаться указатель записи, будет первый открытый индексный файл.

В FoxPro допускается работа сразу с несколькими БД и при этом возможна установка связей между ними. Указатель записей в связанных БД будет двигаться синхронно. БД, в которой указатель движется произвольно, считается старшей, а БД, в которой указатель следует за указателем старшей базы, — младшей или подчиненной. Естественно, в таких базах должны существовать согласованные поля связи. Возможно наличие связей типа 1:1 и 1:M.

Каждый DBF-файл и все соответствующие ему вспомогательные файлы открываются в своей отдельной рабочей области, таким образом одновременно может существовать 25 рабочих областей.

Работа с данными в FoxPro может выполняться следующими способами:

– обработка данных через системное меню FoxPro;

– обработка данных с помощью прикладных программ, созданных программистом;

– обработка данных с помощью программ, созданных средствами генератора приложений.

В FoxPro имеется эффективный язык программирования пользовательских приложений, обладающий мощными командами обработки данных, развитыми диалоговыми средствами, возможностью ускоренного доступа к данным и другими характеристиками языков высокого уровня. Программный код приложения хранится в PRG-файле.

В FoxPro существуют средства создания заготовок программ: генераторы экранов, отчетов и т.д., которые (программы) в дальнейшем можно расширять и дополнять для выполнения поставленных перед разработчиком задач. В Visual FoxPro по сравнению с предыдущими

версиями добавлены новые средства разработки шаблонов пользовательских приложений. В программах FoxPro разрешается иметь те же типы переменных, что и поля, кроме типа MEMO. В FoxPro также разрешается работа с одномерными и двумерными массивами переменных.

В СУБД FoxPro используются различные типы функций: математические, строковые, для работы с датами, преобразования типов и др. В системе предусмотрена возможность использования процедур, которые могут быть как внутренними, так и внешними (в виде отдельных программных файлов либо находится внутри программы).

Важной особенностью FoxPro явилась возможность работы с окнами, каждое окно является как бы автономным экраном системы, что позволяет обеспечить «многослойный» пользовательский интерфейс. Для работы с окнами в FoxPro были добавлены специальные оконные функции. В СУБД FoxPro помимо специальных команд для работы с данными включен ряд команд из языка ANSI SQL для формирования запросов к БД.

Система поддерживает создание исполняемых EXE-модулей программ, создаваемых с помощью Менеджера проектов. Однако для работы созданного в FoxPro EXE-файла на компьютере, где не установлена СУБД, необходимо наличие специального пакета Distribution Kit, входящего в дистрибутив СУБД FoxPro.

10.2.3. СУБД MS Access

Первая версия СУБД MS Access была разработана фирмой Microsoft в 1992 году и не получила широкого распространения из-за отсутствия встроенного средства разработки и ведения баз данных. Первой из версий MS Access, получившей признание разработчиков явилась СУБД MS Access 2.0, поставляемая отдельно от других программных продуктов Microsoft. Более старшие версии, такие, как MS Access 95, MS Access 97, MS Access 2000 и, наконец, MS Access XP, входят в состав соответствующих версий MS Office.

Microsoft Access позволяет управлять всеми сведениями из одного файла базы данных. В рамках этого файла используются следующие объекты:

таблицы для сохранения данных;

запросы для поиска и извлечения только требуемых данных;

формы для просмотра, добавления и изменения данных в таблицах;

отчеты для анализа и печати данных в определенном формате;

макросы;

модули, содержащие код программы на MS Visual Basic;

страницы доступа к данным для просмотра, обновления и анализа данных из базы данных через Интернет или интрасеть.

Внешний вид окна БД MS Access представлен на рис. 10.2.

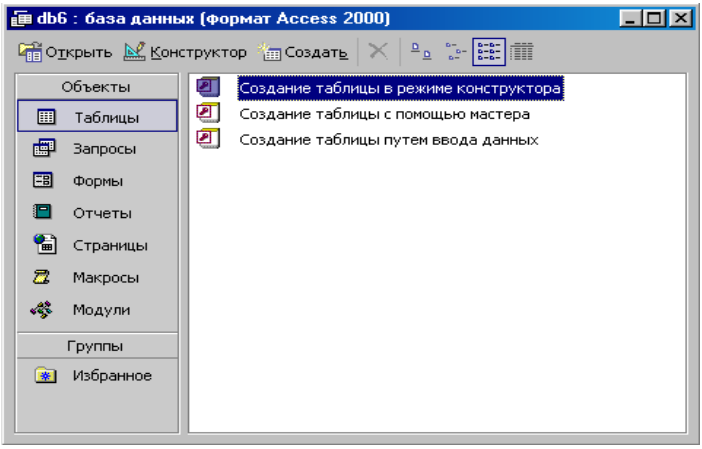

Рис. 10.2. Окно БД MS Access

Ниже представлены характеристики БД в СУБД MS Access XP, взятые из файла описания СУБД, входящего в состав MS Office XP:

размер файла базы данных Microsoft Access (.mdb) — 2 Гбайт за вычетом места, необходимого системным объектам;

число объектов в базе данных — 768;

модули (включая формы и отчеты, свойство Наличие модуля (HasModule) которых имеет значение True) 1 000;

число знаков в имени объекта — 64;

число знаков в пароле — 14;

число знаков в имени пользователя или имени группы — 20;

число одновременно работающих пользователей — 255;

Для простоты просмотра, ввода и изменения данных непосредственно в таблице создается форма. Формы являются типом объектов базы данных, который обычно используется для отображения данных в базе данных. Форму можно также использовать как кнопочную форму, открывающую другие формы или отчеты базы данных, а также как пользовательское диалоговое окно для ввода данных и выполнения действий, определяемых введенными данными. При открытии формы Microsoft Access отбирает данные из одной или более таблиц и выво-

дит их на экран с использованием макета, выбранного в мастере форм или созданного пользователем самостоятельно в режиме конструктора.

Большинство форм являются присоединенными к одной или нескольким таблицам и запросам из базы данных. Источником записей формы являются поля в базовых таблицах и запросах. Форма не должна включать все поля из каждой таблицы или запроса, на основе которых она создается. Присоединенная форма получает данные из базового источника записей. Другие выводящиеся в форме сведения, такие, как заголовок, дата и номера страниц, сохраняются в макете формы.

Формы можно также открывать в режиме сводной таблицы или в режиме диаграммы для анализа данных. В этих режимах пользователи могут динамически изменять макет формы для изменения способа представления данных. Существует возможность упорядочивать заголовки строк и столбцов, а также применять фильтры к полям. При каждом изменении макета сводная форма немедленно выполняет вычисления заново в соответствии с новым расположением данных.

Отчет является эффективным средством представления данных в печатном формате. Имея возможность управлять размером и внешним видом всех элементов отчета, пользователь может отобразить сведения желаемым образом. Большинство отчетов так же, как и формы являются присоединенными к одной или нескольким таблицам и запросам из БД.

Страницы доступа к данным представляют специальный тип вебстраниц, предназначенный для просмотра и работы через Интернет или интрасеть с данными, хранящимися в БД MS Access или в БД MS SQL Server. Страница доступа к данным может также включать данные из других источников, таких, как Microsoft Excel. Использование страниц доступа к данным для ввода данных аналогично использованию форм: пользователь имеет возможность просматривать, вводить, редактировать и удалять данные в базе данных. Однако страницу можно использовать за пределами MS Access, предоставляя пользователям возможность обновлять или просматривать данные через Интернет или интрасеть, для чего пользователям требуется Internet Explorer 5 или более поздняя версия. Со страницами доступа к данным также можно работать в режиме страницы в MS Access. Страницы доступа к данным могут дополнять формы и отчеты, используемые в приложении базы данных.

Макрос в MS Access представляет набор макрокоманд, который создается для автоматизации часто выполняемых задач. Группа макросов позволяет выполнить несколько задач одновременно.

Модули представляют наборы описаний, инструкций и процедур, сохраненных под общим именем для организации программ на языке

MS Visual Basic языке четвертого поколения 4GL. Существуют два основных типа модулей: модули класса и стандартные модули.

Модули форм и модули отчетов являются модулями класса, связанными с определенной формой или отчетом. Они часто содержат процедуры обработки событий, запускаемые в ответ на событие в форме или отчете. Процедуры обработки событий используются для управления поведением формы или отчета и их откликом на события, такие как нажатие кнопки. В Access 97 и более поздних версиях модули класса могут существовать независимо от форм и отчетов. Этот тип модулей класса отображается в окне БД. Модули класса можно использовать для создания описания пользовательского объекта. В Access 95 модуль класса существует только в связи с формой или отчетом.

В стандартных модулях содержаться общие процедуры, не связанные ни с каким объектом, а также часто используемые процедуры, которые могут быть запущены из любого окна БД. Основное различие между стандартным модулем и модулем класса, не связанным с конкретным объектом, заключается в области определения и времени жиз-ни. Значение любой переменной или константы, определенной или су-ществующей в модуле класса, не связанном с конкретным объектом, доступно только во время выполнения этой программы и только из этого объекта.

СУБД MS Access позволяет создавать и сохранять SQL-запросы к БД. Между таблицами БД можно определять связи типа 1:1, 1:M, M:M, возможно также связать таблицу саму с собой. Для отношений, в которых проверяется целостность данных, пользователь имеет возможность указать, следует ли автоматически выполнять для связанных записей операции каскадного обновления и каскадного удаления.

В СУБД MS Access существует возможность присоединения таблиц других БД (Oracle, MS SQL, FoxPro и др.), что позволяет в ее среде генерировать приложения типа клиент-сервер для работы с этими таблицами, возможность импорта и экспорта таблиц в другие БД, а также возможность слияния таблиц и отчетов с другими продуктами, входящими в состав MS Office.

10.3. СУБД третьего поколения (гибридные и ООСУБД), будущее систем управления БД

10.3.1 СУБД Cache

СУБД Oracle, кратко рассмотренную в разделе 9, смело можно отнести к классу гибридных СУБД – здесь успешно сочетаются реляционный и объектно-ориентированный подход. В этом разделе мы по-

знакомимся с СУБД принципиально нового постреляционного класса. При написании данного раздела были использованы материалы опубликованные в [23, 24].

При всех достоинствах современной объектной технологии разработки баз данных имеется несколько препятствий, которые удерживают разработчиков от принятия решения о переходе с реляционной технологии на объектную. Основным останавливающим фактором является значительный объем разработок, опирающихся на реляционные СУБД, поскольку при переходе на объектную технологию необходимо перестраивать не только модель предметной области, но и полностью переписывать все пользовательские приложения, и поэтому возникает вопрос целесообразности такого перехода. СУБД Cache фирмы InterSystems обеспечивает не только реализацию основных возможностей объектно-ориентированной технологии, но и позволяет во многом облегчить переход с реляционной технологии на объектную, а также может выступать в роле шлюза к реляционным базам данных.

Отличительной особенностью СУБД Cache является независимость хранения данных от способа их представления. Это реализуется с помощью так называемой единой архитектуры данных Cache. В рамках данной архитектуры существует единое описание объектов и таблиц, отображаемых непосредственно в многомерные структуры ядра базы данных, ориентированного на обработку транзакций. Как только определяется класс объектов, Cache автоматически генерирует реляционное представление данных этого класса. Подобным же образом, как только в словарь данных поступает DDL-описание на языке SQL, Cache автоматически генерирует реляционное и объектное описание данных. При этом все описания ведутся согласованно, но все операции по редактированию проводятся только с одним описанием данных. Это позволяет сократить время разработки. Одновременно улучшается совместимость со старыми SQL-ориентированными приложениями.

На рынке высокопроизводительных СУБД Cache позиционируется как eDBMS, т.е. как СУБД, ориентированная на работу в сетях Internet/Intranet. Поэтому при установке проверяется наличие web-сервера и производится автоматическое конфигурирование подсистемы. В версии Cache 4.0. реализована технология создания динамических web-приложений Cache Server Pages (CSP. Наряду с этим, системная библиотека Net предоставляет классы, реализующие протоколы SMTP, POP3, HTTP, FTP и др.

На рис. 10.3 представлена архитектура СУБД Cache.

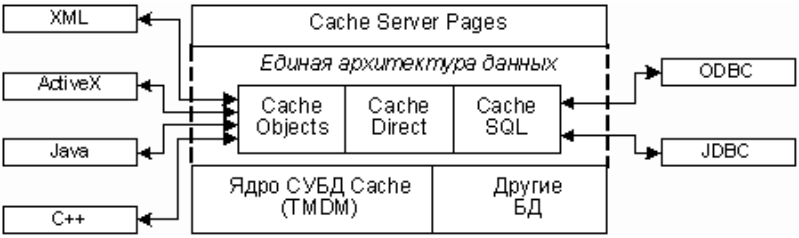

Рис. 10.3 Архитектура системы Cache

Основные компоненты СУБД Cache:

1. *TMDM.* Многомерное ядро системы, ориентированное на работу с транзакциями.

Данные в Cache хранятся в виде разреженных массивов, носящих название глобалей. Количество индексов массива может быть произвольным, что позволяет описывать и хранить структуры данных произвольного уровня сложности. Индексы глобалей могут быть любого литерального типа данных.

Применение разреженных массивов позволяет оптимизировать использование объема жесткого диска и сократить время, требуемое на выполнение операций ввода/вывода и извлечение данных.

В СУБД Cache реализована развитая технология обработки транзакций и разрешения конфликтов. Блокировка данных производится на логическом уровне. Это позволяет учитывать особенность многих транзакций, производящих изменения небольшого объема информации. Кроме этого, в Cache реализованы атомарные операции добавления и удаления без проведения блокировки, в частности, это применяется для счетчика идентификаторов объектов.

2. *Сервер Cache Objects*. Представление многомерных структур данных ядра системы в виде объектов, инкапсулирующих как данные, так и методы их обработки.

Объектная модель Cache соответствует объектной модели стандарта ODMG (Object Data Management Group). В соответствии со стандартом ODMG каждый экземпляр объекта в Cache должен быть определенного типа. Поведение объекта определяется операциями (методами), а состояние объекта – значениями его свойств. Свойства и операции составляют характеристики типа. Тип определяется одним интерфейсом, которому может соответствовать одна или большее число реализаций.

Объектная модель Cache представлена на рис. 10.4.

Рис. 10.4 Объектная модель Cache

В соответствии со стандартом в Cache реализовано два типа классов:

– классы типов данных (литералы);

– классы объектов (объекты).

Классы типов данных определяют допустимые значения констант (литералов) и позволяют их контролировать. Литерал не имеют дополнительной идентификации, кроме своего значения, в то время как объекты имеют еще и уникальную идентификацию.

Классы типов данных подразделяются на два подкласса типов:

– атомарные;

– структурированные.

Атомарными литеральными типами в Cache являются традиционные скалярные типы данных (String, Integer, Float, Date и др.). В Cache реализованы две структуры классов типов данных – список и массив. Каждый литерал уникально идентифицируется индексом в массиве или порядковым номером в списке.

Различают два подтипа классов объектов – зарегистрированные и незарегистрированные. Зарегистрированные классы обладают предопределенным поведением, т.е. набором методов, наследуемых из системного класса RegisteredObject и отвечающих за создание новых объектов и за управление размещением объектов в памяти. Незарегистрированные классы не обладают предопределенным поведением, разработка функций (методов) класса целиком и полностью возлагается на разработчика.

Зарегистрированные классы могут быть двух типов – сериализуемые и хранимые. Сериализуемые классы наследуют свое поведение от системного класса SerialObject. Основной особенностью

хранения сериализуемого класса является то, что объекты сериализуемых классов существуют в памяти как независимые экземпляры, однако могут быть сохранены в базе данных, только будучи встроенными в другой класс.

Хранимые классы наследуют свое поведение от системного класса Persistent, который предоставляет своим наследникам обширный набор функций, включающий: создание объекта, подкачку объекта из БД в память, удаление объекта и т.п.

Объектная модель Cache в полном объеме поддерживает все основные концепции объектной технологии:

Наследование. Объектная модель Cache позволяет наследовать классы от произвольного количества родительских классов.

Полиморфизм. Объектная модель Cache позволяет создавать приложения целиком и полностью независимыми от внутренней реализации методов объекта.

Инкапсуляция. Объектная модель Cache обеспечивает сокрытие отдельных деталей внутреннего устройства классов от внешних по отношению к нему объектов или пользователей. Разделяют интерфейсную часть класса и конкретную реализацию. Интерфейсная часть необходима для взаимодействия с любыми другими объектами. Реализация же скрывает особенности реализации класса, т.е. все, что не относиться к интерфейсной части.

Хранимость. Система Cache поддерживает несколько видов хранения объектов: автоматическое хранение в многомерной базе данных Cache; хранение в любых структурах, определенных пользователем; хранение в таблицах внешних реляционных баз данных, доступных через шлюз Cache SQL Gateway.

3. *Сервер Cache SQL*. Представление многомерных структур данных в виде реляционных таблиц.

Наряду с реализацией в полном объеме основных принципов объектной технологии в СУБД Cache поддерживается структурированный язык запросов SQL. Это, как уже говорилось, обеспечивает выполнение запросов по стандарту, поддерживаемому многими инструментальными средствами. Кроме этого, с помощью единой архитектуры данных Cache возможно автоматическое преобразование описаний реляционных таблиц в классы объектов. При поступлении на сервер Cache SQL DDL-описания реляционной таблицы, Cache автоматически преобразует DDL-описание в свою внутреннюю форму и сохраняет полученную структуру в словаре данных. С помощью поставляемых в стандартной комплектации Java- или ODBC-драйверов возможен также и импорт данных из

реляционных таблиц в многомерные структуры ядра Cache. Это позволяет Cache работать с данными как в виде реляционных таблиц, так и в виде классов объектов. Таким образом, при переходе с реляционной на объектную технологию разработка не начинается с «нуля» – многое делается Cache автоматически.

4. *Сервер прямого доступа (Cache Direct)*. Предоставление прямого доступа к многомерным структурам данных ядра системы.

С помощью сервера Cache Direct разработчик получает доступ к многомерным структурам ядра системы. Встроенный в СУБД Cache язык программирования COS предоставляет ряд функций для работы с массивами данных, или глобалями, составляющими ядро системы.

Использование прямого доступа к данным позволяет оптимизировать время доступа к данным. Для прямого доступа и работы с многомерными структурами ядра системы можно воспользоваться утилитой эмуляции ASCII-терминала Cache Terminal, которая обычно используется для целей обучения языку COS и тестирования работы терминального приложения

В стандартной поставке системы разработчику предлагается два средства администрирования Cache:

– *Configuration Manager;*

– *Control Panel.*

С помощью *Configuration Manager* можно выполнить следующие функции администрирования:

– создать новую БД, удалить или изменить настройки существующей БД. С точки зрения физического хранения, БД Cache – это бинарный файл CACHE.DAT. Для каждой БД создается свой файл CACHE.DAT в отдельной директории;

– определить область (Namespace) для существующей БД, под которой в Cache понимается логическая карта, на которой указаны имена многомерных массивов – глобалей и программ файла CACHE.DAT, включая имена каталога-директории и сервера данных для этого файла. При обращении к глобалям используется имя области;

– определить CSP-приложение. Для использования CSP-приложений необходимо определить виртуальную директорию на web-сервере, физическую директорию хранения CSP-приложений, а также несколько специфических для CSP настроек, таких как, к примеру, класс-предок для CSP приложений (по умолчанию принимается системный класс CSP.Page);

– определить сетевое окружение Cache. В Cache реализован собственный протокол для работы БД в распределенной среде, носящий название DCP (Distributed Cache Protocol). С помощью интерфейсов

Configuration Manager можно определить источники данных, а также связи между различными компонентами сети;

– настроить систему Cache. Разработчику предоставляется возможность конфигурирования различных компонентов Cache, таких как параметры журналирования, настройки теневых серверов, параметры сервера лицензий, параметры Cache-процессов и другие.

Утилита *Control Panel* предоставляет схожий набор функций администрирования, но также позволяет осуществлять:

– управление процессами Cache.

– настройка параметров защиты глобалей, таких как разрешение на редактирование/создание/чтение глобалей различными группами пользователей.

– определение пользователей системы с присваиванием им имени пользователя, пароля и определение параметров доступа.

– просмотр файлов журнала. Журналирование в Cache выполняется на уровне глобалей.

– определение теневых серверов системы.

– создание резервных копий баз данных.

В СУБД Cache реализован собственный язык программирования Cache Object Script (COS). COS – это расширенная и переработанная версия языка программирования M (ANSI MUMPS).

В первую очередь, COS предназначен для написания исходного кода методов класса. Кроме этого, в Cache вводится понятие Cache-программы. Cache-программа не является составной частью классов и предназначена для написания прикладного программного обеспечения для текстовых терминальных систем.

Основой концепции серверных страниц Cache является автоматическое создание по запросу пользователя web-страниц, содержащих требуемую информацию из БД Cache. Вся бизнес-логика CSP-приложений выполняется в непосредственной близости к хранилищу данных Cache, таким образом сокращается объем данных, которыми обмениваются web-сервер и сервер БД Cache, что приводит к выигрышу в производительности по сравнению с другими технологиями создания web-приложений. Для еще большего увеличения производительности CSP приложений при обмене данными между сервером Cache и web-сервером используются высокоскоростные API-интерфейсы.

Серверные страницы Cache представляют собой HTML-файлы, содержащие дополнительные теги Cache (Cache Application Tags или CATs). Для создания CSP приложений можно воспользоваться стандартными средствами разработки HTML-страниц (Cache

предоставляет add-in модуль для полной интеграции с Macromedia DreamWeaver) или, на крайний случай, обыкновенным текстовым редактором.

СУБД Cache поддерживает множество национальных языков. Кроме поддержки языков, специальная утилита CNLS позволяет создавать собственные таблицы трансляции из одного набора символов в другой, задавать различные способы вывода непечатных символов и предоставляет ряд других возможностей. При инсталляции под ОС Windows Cache автоматически определяет региональные настройки операционной системы и устанавливает соответствующую схему локализации. Также предоставляется возможность установки Unicode-версии (16bit) Cache.

Минимальные требования к аппаратному обеспечению для работы под ОС Windows:

– процессор Intel Pentium;

– ОЗУ – 64 Мбайт (минимум);

– 100 Мбайт свободного места на диске;

– сконфигурированный протокол TCP/IP с фиксированным IP-адресом.

Кроме описанных интерфейсов, Cache предоставляет ODBC- и JDBC-драйверы для представления данных из СУБД Cache в виде реляционных таблиц и работы с ними.

СУБД Cache предоставляет стандартные ActiveX-компоненты, которыми можно воспользоваться при создании пользовательского приложения в таких средствах разработки, как Visual Basic. Кроме этого, предоставляется мастер создания форм Cache Form Wizard для облегчения разработки пользовательских форм в среде Visual Basic.

Кроме всего перечисленного, в следующей версии Cache планируется обеспечить поддержку XML – общепринятого стандарта для обмена данными между различными платформами и SOAP-протокола для удаленного вызова функций.

10.3.2 Перспективы развития СУБД

Рассмотрев более-менее подробно некоторые СУБД различных поколений отметим перспективы развития СУБД.

Перспективы гибридных и расширенных СУБД [6]:

1) системы управления реляционными базами данных будут поддерживать такие элементы ООП, как абстрактные типы данных, расширятся возможности использования хранимых процедур при вза-

имодействии с объектными типами, наследование и инкапсуляция будут использоваться на должном уровне в таких системах;

2) ОО-приложения на основе стандарта JDBC будут осуществлять доступ к реляционным БД;

3) SQL будет содержать возможность построения ОО-конструкций.

Перспективы ООСУБД:

1) серверы, соответствующие стандарту CORBA (стандарт общей архитектуры брокера объектных запросов, позволяющий интегрировать различные ООСУБД), обеспечат предоставление возможности ООБД многим классам приложений;

2) должен быть отлажен механизм защиты данных в ООБД;

3) будут унифицированы механизмы реализации моделей транзакций, допускающие создание неоднородных сред ООБД;

4) архитектура CORBA будет расширяться, включая более мощные возможности по реализации ОО-подхода в области БД.

Контрольные вопросы

1. Перечислите и охарактеризуйте СУБД 1-го и 2-го поколения.
2. Поясните различие СУБД, функционирующих в архитектуре клиент-сервер, и файл-серверных СУБД.
3. Перечислите и охарактеризуйте основные объекты СУБД MS ACCESS.
4. Опишите основные свойства СУБД Cache

Список используемой литературы

1. Кузнецов С.Д. Основы современных баз данных // Системы управления базами данных. — 1995. — №№ 1–5.

2. Чудинов И.Л. Организация баз данных: Учебное пособие. — Томск: ТУСУР, 2000. — 89 с.

3. Матрин Дж. Организация баз данных в вычислительных системах. — М: Мир, 1980.

4. U. Dayal et al., “Third Generation TP Monitors: A Database Challenge”, Proceeding of the 1993 ACM SIGMOD, 394.

5. J. Melton and A. Simon, Understanding the New SQL: A Complete Guide (San Francisco: Morgan Kaufmann Publishers, 1992), 39-40.

6. Саймон А.Р. Стратегические технологии баз данных: менеджмент на 2000 год. — М.: Финансы и статистика, 1999. — 479 с.

7. Атре Ш. Структурный подход к организации баз данных. — М.: Финансы и статистика, 1983.

8. E. F. Codd. “A Relational Model of Data for Large Shared Data Banks”, Communications of the ACM (June 1970).

9. Кодд Э.Ф. Реляционная модель данных для больших, совместно используемых банков данных. // Системы управления базами данных. — 1995. — № 1, с. 145-160.

10. Дейт, К.Дж. Введение в системы баз данных. — Киев, М.: Диалектика, 1998. — 784 с.

11. Мишель Пуле. Четыре грани целостности // http://www.osp.ru/win2000/sql/2000/01/010.htm

12. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. – М.: Финансы и статистика, 1998.

13. CASE-Технологии // http://case-tech.h1.ru

14. Калянов Г. Российский рынок CASE-средств // PC Week, 1998. – № 3.

15. Семизельникова О.А. Исследование возможностей CASE-технологии при создании интеллектуальных систем.// http://www.inftech.webservis.ru.

16. 16. Сборник временных норм на работы по ведению Государственного мониторинга геологической среды, информационной деятельности, цифровому картографированию/ http://www.tgm.ru/l2/otch.html

17. Ульман Дж. Основы систем баз данных. — М.: Финансы и статистика, 1983.

18. Пейдж, Вильям, Дж., и др. Использование Oracle 8/8i. — М.: Издательский дом «Вильямс», 2000. — 1024 с.

19. Системы баз данных третьего поколения: Манифест / Комитет по развитию функциональных возможностей СУБД // Системы управления базами данных. — 1995. —№ 2.

20. М. Аткинсон, Ф. Бансилон, Д. ДеВитт, К. Диттрих, Д. Майер, С. Здоник. Манифест систем объектно-ориентированных баз данных // Системы управления базами данных. — 1995. — № 4.

21. Марк Ривкин, новые возможности oracle 9.2 // «Открытые системы», 2002 №11.

22. Попов А.А. Программирование в среде СУБД FoxPro 2.0. Построение систем обработки данных. — Киев: ТОО "ВЕК"; М.: Радио и связь, 1995. — 352 с.

23. В.Кирстен, М.Ирингер и др. «СУБД Cache'. Объектно-ориентированная разработка приложений», Питер, 2001.

24. Сиротюк О. Постреляционная СУБД Cache http://www.cit-forum.ru/database/articles/subdcache.shtml