Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

ФАКУЛЬТЕТ ДИСТАНЦИОННОГО ОБУЧЕНИЯ (ФДО)

П. В. Сенченко

ОРГАНИЗАЦИЯ БАЗ ДАННЫХ

Учебное пособие

Томск
2015

УДК 004.65(075.8)
ББК 32.973.233-018.2я73
С 316

Рецензенты:
Тарасенко В. Ф., докт. техн. наук, профессор кафедры теоретической кибернетики Национального исследовательского Томского государственного университета;
Перемитина Т. О. канд. техн. наук, доцент кафедры автоматизации обработки информации ТУСУР.

Сенченко П. В.

С 316 Организация баз данных : учебное пособие / П. В. Сенченко. — Томск : ФДО, ТУСУР, 2015. — 170 с.

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.

Ориентировано на студентов направлений подготовки бакалавров «Государственное и муниципальное управление», «Бизнес-информатика», «Программная инженерия», а также студентов родственных специальностей и разработчиков баз данных.

УДК 004.65(075.8)
ББК 32.973.233-018.2я73

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Учебное пособие составлено в соответствии с требованиями основных образовательных программ Федеральных государственных образовательных стандартов подготовки бакалавров по направлениям «Программная инженерия», «Бизнес-информатика», «Государственное и муниципальное управление» по дисциплине «Базы данных».

Изучение материала, изложенного в данном пособии, должно не только помочь в освоении новых знаний будущим специалистам в области разработки информационных технологий, но и сформировать общий базис по организации баз данных (БД) квалифицированным специалистам в различных областях деятельности, связанных с использованием прикладных программных продуктов.

Целью изучения данного курса является овладение способами организации и методами проектирования баз данных, а также технологией их использования в системах обработки информации и управления.

Учебное пособие содержит следующие основные разделы 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;
- современные методики проектирования баз данных.

Для эффективного освоения материала, изложенного в учебном пособии, студентам целесообразно знать: основы информатики, общие представления о разработке информационных технологий, подходы к обработке данных, основы теории множеств и применения теоретико-множественных операций.

В результате изучения дисциплин с использованием данного пособия студент приобретает знания, необходимые для эффективного проектирования сложно структурированных баз данных для любых предметных областей с использованием методов нормализации и моделирования данных. В ходе изучения практической составляющей пособия студенты получают возможность создания баз данных и простых информационных технологий в среде современных систем управления базами данных.

Знание организации БД позволит на профессиональном уровне овладеть навыками работы в конкретных прикладных системах по ведению и обработке информации.

Соглашения, принятые в книге

Для улучшения восприятия материала в данной книге используются пиктограммы и специальное выделение важной информации.

Этот блок означает определение или новое понятие.

Этот блок означает внимание. Здесь выделена важная информация, требующая акцента на ней. Автор здесь может поделиться с читателем опытом, чтобы помочь избежать некоторых ошибок.

Эта пиктограмма означает совет. В данном блоке можно указать более простые или иные способы выполнения определенной задачи. Совет может касаться практического применения только что изученного или содержать указания на то, как немного повысить эффективность и значительно упростить выполнение некоторых задач.

Этот блок означает теорему.

Пример

Этот блок означает пример. В данном блоке автор может привести практический пример для пояснения и разбора основных моментов, отраженных в теоретическом материале.

Контрольные вопросы по главе

Глава 1

ОБОСНОВАНИЕ КОНЦЕПЦИИ БАЗ ДАННЫХ

1.1 История и направления развития вычислительной техники

История развития технических средств, способных облегчить умственный труд человека, насчитывает несколько сотен лет. Так, одним из первых устройств для выполнения арифметических вычислений принято считать счетную доску абак, которую использовали еще в Древнем Вавилоне в третьем тысячелетии до нашей эры. В России абак или более известные деревянные русские счеты появились намного позже, в XV веке нашей эры. В XVII веке немецкий ученый Вильгельм Шиккард разработал первый механический калькулятор «Считающие часы», с помощью которого можно было выполнить четыре основных арифметических действия. Немного позднее были изобретены счетные машины Блеза Паскаля и Готфрида Вильгельма Лейбница.

Первый серийный арифмометр был создан Шарлем Ксавье Томасом де Кольмаром в 1820 году. Подобные механические калькуляторы применялись вплоть до 70-х годов XX века. В начале XIX века Жозеф Мари Жаккар разработал работающей с использованием перфокарт ткацкий станок. Для того чтобы изменить узор на ткани, необходимо было просто заменить набор перфокарт. Подобная технология в дальнейшем использовалась в разработке аналитических машинах Чарльза Бэббиджа. Вычислительные устройства, созданные на основе применения перфокарт, а позднее и перфолент, использовались повсеместно до 80-х годов прошлого века.

В 1904 году в Российской империи русским инженером Алексеем Николаевичем Крыловым была изобретена вычислительная машина, способная решать дифференциальные уравнения. В первой половине XX века уже стало обыденным использование механических вычислителей в труде математиков, бухгалтеров и финансовых работников. Так, в Советском Союзе был создан арифмометр «Феликс», выпускавшийся с 1929 по 1978 годы, фактически это был первый отечественный механический калькулятор.

С началом эпохи электричества начинался новый виток в развитии вычислительных машин. В 1890 году Германом Холлеритом была создана электромеханическая машина, способная выполнять различные математические расчеты, оперируя информацией, зафиксированной на перфокартах, обеспечивая вывод данных на бумажные носители. Такая машина, получившая название «табулятор», использовалась в обработке переписи населения США в 1890 году. Холлеритом была организована фирма, основной задачей которой являлось производство табуляторов для различных сфер деятельности (партия таких машин была закуплена правительством Российской империи). Фирма Холлерита динамично развивалась, а с 1924 года получила широко известное название IBM.

В 1935 году в СССР на Первом Государственном заводе счетных и счетно-аналитических машин была создана первая в нашей стране электродинамическая счетно-аналитическая машина «Табулятор Т1». В 1941 году немецким инженером Конрадом Цузе была спроектирована и воплощена в жизнь Z3 — первая полноценная вычислительная машина, созданная на основе обыкновенных телефонных реле. Для этой разработки были характерны основные свойства современных компьютеров — существовали возможности по программному управлению и программированию в двоичном коде с плавающей запятой.

В середине XX века в процессе развития и совершенствования вычислительной техники сформировалось два основных направления ее использования. Первое направление (40-е–50-е годы) характеризовалось широкомасштабным применением электронно-вычислительной техники для выполнения сложных математических расчетов, которые трудоемко или вообще невозможно производить вручную.

Так, в 1943 году в США была создана первая за океаном вычислительная машина Марк I, основное назначение которой заключалось в возможности проведения баллистических расчетов для Министерства обороны США. 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. Одним из недостатков первого направления являлась невозможность повторного использования исходных данных.

Дополнительным толчком в развитии вычислительной техники явилось изобретение в 1960 году интегральной схемы. Для этого периода характерно второе направление (60-е годы), непосредственно касающееся темы нашего курса, — это использование средств вычислительной техники для разработки и функционирования автоматизированных информационных систем.

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. Примерами информационных систем являются библиотечные системы, системы бухгалтерского и финансового учета, банковские системы, системы резервирования авиационных или железнодорожных билетов, системы складского учета и т. д.

Естественно, что второе направление возникло несколько позже первого. Это связано с тем, что на начальном этапе развития вычислительной техники компьютеры обладали ограниченными возможностями в области 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.

Для первоначального периода создания средств вычислительной техники характерно использование нескольких типов носителей информации во внешней памяти: магнитных лент, перфолент, перфокарт и барабанов. При этом перфоленты и перфокарты были способны 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 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, наличия 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 ее развития. Эти устройства внешней памяти обладали существенно большей емкостью, чем магнитные барабаны, обеспечивая 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 прошлое.

С изобретением 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 приходилось 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 файлов.

*С точки зрения прикладной программы **файл** — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные.*

Работа с файлами невозможна без наличия операционной системы.

***Операционная система** — это базовый комплекс компьютерных программ, обеспечивающий управление аппаратными средствами компьютера, работу с файлами, ввод и вывод информации, а также выполнение прикладных программ и утилит.*

Специфические правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами — прикладной программы, обеспечивающей возможность формирования файла, записи его на носитель информации и открытия для дальнейших действий.

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. 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. С точки зрения системы управления файлами объектные файлы и библиотеки дополнительных функций файловой системы обладают очень простой структурой, представляющей собой последовательность записей или байтов [1]. Система программирования накладывает на эту структуру более сложную и специфичную для данной системы структуру объектного модуля. Логическая структура объектного модуля неизвестна файловой системе и поддерживается средствами конкретного языка программирования.

Системы управления файлами в большинстве своем обычно обеспечивают формирование и обработку слабоструктурированной информации, оставляя дальнейшую структуризацию непосредственно прикладным программам. Первые ин-

формационные системы создавались на основе и при помощи простейших систем управления файлами.

1.3 Основные понятия СУБД

При разработке информационной системы определенной предметной области основным фактором является обеспечение надежного хранения и обработки постоянно существующей информации, при этом необходимо обеспечить выполнение дополнительных функций, не всегда свойственных простым файловым системам.

*Под **предметной областью** будем понимать любой тип организационной структуры, например банк, университет, завод, больницу.*

Структура информации в информационных системах 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.

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

Основным недостатком позадачного подхода являлось дублирование исходных данных в различных файлах, что приводило к избыточности хранимой информации, а также к возникновению различных коллизий, проявляющихся в процессе обновления данных.

Можно предположить, что при работе 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 информационной системы (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 систем управления базами данных (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.

Пусть необходимо разработать информационную систему, в которой ведутся сведения о студентах вуза. Такая система должна обеспечивать выполнение следующих функций:

- выдачу по запросу пользователя списка студентов по группам;
- поддержку возможности перевода студента из одной группы в другую;
- прием абитуриентов;
- отчисление студентов.

При этом для каждой группы должна поддерживаться возможность получения информации о кураторе группы, численности студентов, среднем размере стипендии студентов группы за определенный временной интервал и т. д.

Разработать такую информационную систему можно с применением практически любого языка программирования, используя при этом для хранения данных один потоковый текстовый файл. Расширять базовые возможности системы можно за счет специальной библиотеки функций.

Поскольку в качестве минимальной информационной единицы в данной предметной области можно выбрать студента, в этом файле содержится одна структурная единица (запись) для каждого студента, которая может быть представлена в следующей последовательности, отражающей основные сведения о студентах: ФИО студента (ФИО); номер студенческого билета (№_СТУД_БИЛЕТА); дата рождения (ДАТА_РОЖД); номер группы (№_ГРУППЫ); размер стипендии (СТИП); балл ЕГЭ (ЕГЭ). Используя один файл для хранения информации, необходимо учитывать, что эта же запись должна содержать сведения и по группе, в которой обучается студент, — ФИО куратора (КУРАТОР); средний балл ЕГЭ (СР_ЕГЭ); количество студентов (КОЛ_СТУД).

Поскольку в исходном файле хранится информация по разным студентам, то для решения поставленных функциональных задач в такой системе необходимо обеспечить доступ к конкретной записи по заранее заданным указателям (ключам) — уникальным в разных записях. В нашем случае в качестве такого ключа может выступать №_СТУД_БИЛЕТА. Кроме того, должна обеспечиваться возможность выбора всех записей с общим значением №_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 информационной системы требует создания достаточно сложной функциональной надстройки для организации доступа к файлу. Кроме этого, прослеживается явная избыточность хранения данных — для каждого студента группы повторяется ФИО куратора, средний балл ЕГЭ, количество студентов.

Одним из способов реструктуризации хранения информации в такой системе является возможность поддерживать два информационных файла: СТУДЕНТЫ и ГРУППЫ. Первый файл должен содержать сведения, касающиеся непосредственно студентов: ФИО, №_СТУД_БИЛЕТА, СТИП, №_ГРУППЫ, а второй — НОМЕР_ГРУППЫ, КУРАТОР, СР_ЕГЭ, КОЛ_СТУД. В этом случае мы ликвиди-

руем избыточность данных — каждый из файлов будет содержать только уникальную информацию. Однако даже при такой модификации информационная система должна обладать некоторыми дополнительными особенностями, сближающими ее с СУБД.

Прежде всего, в системе должно быть указано, что она работает с двумя информационно связанными файлами, и описаны структура и смысл каждого поля (например, что №_ГРУППЫ в файле СТУДЕНТЫ и №_ГРУППЫ в файле ГРУППЫ означают одно и то же). Кроме того, необходимо учитывать, что в ряде случаев изменение информации в одном файле должно автоматически вызывать модификацию во втором файле, чтобы содержимое этих файлов было ***согласованным***. Например, в случае изменения номера группы в файле ГРУППЫ все соответствующие номера групп в файле СТУДЕНТЫ должны быть изменены на новое значение.

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 недостаточный признак СУБД. 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.

Кроме этого, в нашем примере средствами информационной системы трудоемко реализовывать запросы на выдачу информации, например «выдать ФИО куратора группы, в которой обучается Раевский Александр Иванович». Было бы гораздо проще, если бы система позволяла сформулировать этот запрос на близком пользователям языке. Такие языки называются языками запросов к базам данных. Например, на языке SQL (Structured Query Language) такой запрос можно было бы реализовать средствами СУБД и выразить в форме:

```
SELECT Группы.Куратор
FROM Студенты, Группы
WHERE ФИО = "Раевский Александр Иванович"
AND Студенты. №_Группы = Группы.№_Группы;
```

При наличии возможности обработки подобных запросов СУБД позволит не задумываться о том, как будет выполняться этот запрос, а автоматически выполнит просмотр файлов СТУДЕНТЫ и ГРУППЫ и выдаст соответствующий результат.

Поддержка корректности состояния базы данных во время сбоев в момент вставки или удаления данных не является обязательной функцией информационных систем. Данная функция также может быть реализована средствами СУБД.

Также следует предусмотреть одновременную работу с информационной системой нескольких пользователей. Для обеспечения корректности на все время

модификации любого из двух файлов доступ других пользователей к этому файлу необходимо каким-то образом ограничить или полностью блокировать. Таким образом, занесение в файл сведений о новом студенте Данилове Олеге Владимировиче существенно замедлит процесс получения информации о студенте Карасеве Алексее Александровиче, даже если они будут числиться в разных группах. СУБД при этом способна обеспечить гораздо более тонкую синхронизацию параллельного доступа к данным.

Кроме того, существует возможность значительно упростить структуру файла ГРУППЫ, для чего можно удалить средний балл ЕГЭ (СР_ЕГЭ), количество студентов (КОЛ_СТУД) из данного файла. При этом СУБД необходимо дополнить возможностью обработки простых статистических функций, таких как AVG — вычисление среднего значения из набора предложенных и SUM — вычисление суммы.

Таким образом, система управления базами данных способна решить множество задач, которые затруднительно или вообще невозможно решить собственными силами информационных систем и прикладных программ.

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

1) автономное безызбыточное 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 могут быть сформулированы следующим образом:

- база данных есть отображение информационной модели предметной области;
- однократный ввод первичной информации;
- обеспечение различных уровней защиты данных (защита от катастрофического разрушения, защита от несанкционированного доступа, ограничение целостности);
- реорганизация (развитие) БД по мере необходимости с минимальным влиянием на действующие программы [2].

Эти положения легли в основу большинства существующих определений БД и СУБД. Дж. Мартин в 1980 году сформулировал понятия БД и СУБД.

База данных *есть совокупность 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 набор специальных программных приложений, предназначенных для обеспечения эффективного доступа к базе данных, используемый для предоставления только необходимой информации, обеспечения независимости от возможных изменений в структуре той части базы данных, которую не обрабатывает программа.*

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 банками данных, а затем — базами данных.

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 период разработки первых СУБД 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, 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;
2) объединением последних в комплексы (подсистемы) с постоянным их совершенствованием, включающим расширение состава задач и ориентирование на широкий круг конечных пользователей;
3) противоречием между позадачным подходом в использовании исходных данных и требованием их эффективной актуализации;
4) стремлением отобразить в системе хранимых данных информационную модель определенной предметной области.

1.4 Функции СУБД

За время эволюции СУБД был определен не только круг решаемых задач, но и перечень основных функций, поддержка которых в системе необходима для их решения. В каждой конкретной СУБД имеется 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 [1]:

1) управления данными во внешней и оперативной памяти компьютера;
2) управления транзакциями;
3) журнализации изменений БД;
4) поддержки языков доступа к данным;
5) обеспечения безопасности базы данных.

Управление данными во внешней (на жестком диске) и оперативной памяти компьютера

Данная функция 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 — управление 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 транзакциями

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 СТУДЕНТЫ и ГРУППЫ (см. подраздел 1.3), то единственным способом не нарушить целостность БД при выполнении операции изменения номера группы является объединение элементарных операций над файлами СТУДЕНТЫ и ГРУППЫ в одну транзакцию. Таким образом, поддержание механизма транзакций является обязательным условием даже для однопользовательских СУБД, но особенно важным и необходимым — 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*) [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 его целостным после своего завершения, что делает очень удобным использование транзакции как единицы активности пользователя по отношению к БД. Механизм 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 до тех пор, пока производимые обновления не будут зафиксированы. Система должна обеспечить для каждой транзакции иллюзию того, что она выполняется изолированно, 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 *сериализацией параллельно выполняющихся транзакций* понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения [1].

Сериальный план выполнения смеси транзакций — *это такой план, который приводит к сериализации транзакций.*

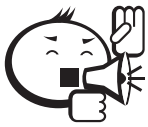
Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

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 сохраняться 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. Подробная информация о моделях транзакций и системах управления транзакциями изложена в [6].

Журнализация изменений БД



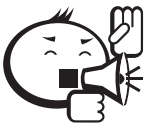
Главным критерием при выборе СУБД является ***надежность хранения информации***.

Здесь под этим термином будем понимать возможность СУБД восстановить последнее согласованное состояние БД после возникновения сбоя.

Различают два вида сбоев: программный и аппаратный. Эти сбои, в свою очередь, можно разделить на мягкие сбои, в результате которых происходит нарушение работоспособности СУБД, которое не влечет потерю данных, и жесткие сбои, при которых возможна частичная или полная потеря информации. Под программным сбоем, например, можно подразумевать аварийное завершение работы СУБД в результате действия вирусных программ. Под аппаратным — перепад напряжения, который может привести к выходу из строя жесткого диска. 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, — вести журнал изменений БД.

Журнал изменений БД *— это часть БД, в которую поступает информация обо всех изменениях базы данных.*

В некоторых СУБД журнал недоступен пользователям СУБД и может храниться в нескольких экземплярах на разных носителях. Идеология формирования журнала в большинстве СУБД основывается на необходимости соблюдения принципа упреждающей записи об изменениях БД в журнал WAL (*Write Ahead Log*), т. е. информация об изменениях данных в БД в журнале должна появиться до того, как произойдут эти изменения. Если в СУБД корректно ведется журнал изменений БД, то с его помощью можно восстановить базу данных 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 обеспечение такого состояния внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и 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, находящуюся в согласованном состоянии. Можно дать две основные рекомендации по ведению архивных копий БД.

1. Архивная копия БД должна храниться на другом физическом носителе.
2. Архивная копия БД должна создаваться регламентно — по итогам определенного периода наполнения БД, с заданной периодичностью, перед внесением изменений в структуру БД и т. д.

Принцип восстановления БД состоит в том, что по архивной копии и следуя журналу, должны быть отработаны все завершенные транзакции.

Поддержка языков доступа БД

Основной функцией любой СУБД, помимо хранения данных, является возможность оперировать этими данными. Для работы с БД используются синтаксические конструкции, в целом называемые языками баз данных. В большинстве ранних СУБД изначально поддерживалось несколько типов языков. Собственно, выделялось 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 создания собственно базы данных и ее объектов и заканчивая выводом структурированной информации в ответ на запрос, реализованный на этом языке. Одним из таких языков, используемым в большинстве реляционных СУБД, является язык SQL. Язык SQL позволяет выполнять функции языков SDL и DML, с его помощью можно генерировать схему БД и манипулировать данными. С помощью языка SQL можно создавать сложные конструкции, определяя, в том числе, и ограничения целостности БД. Кроме SDL и DML в состав языка SQL также включен ряд дополнительных операторов, например операторы языка определения доступа к данным (рис. 1.1).

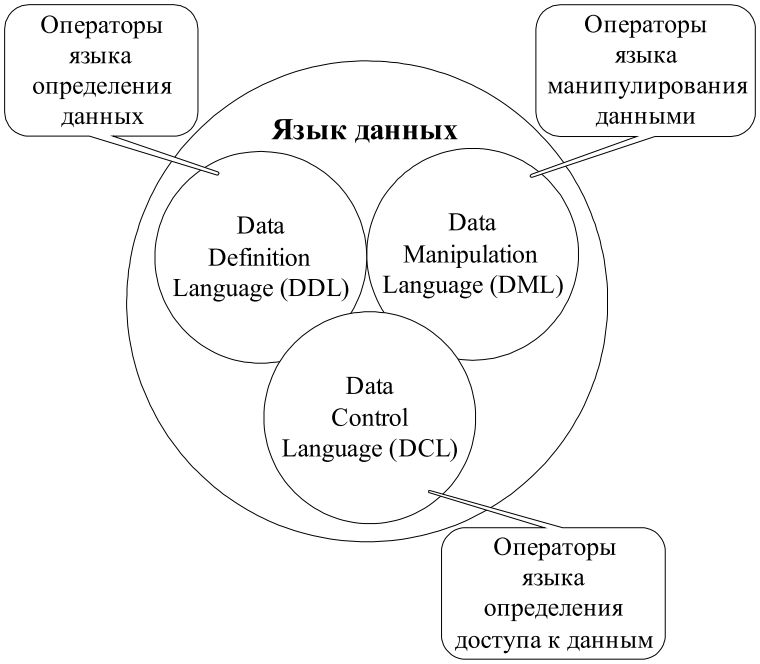


Рис. 1.1 – Состав языка данных

Подробно стандарты и функции языка SQL и других языков манипулирования данными рассмотрены в главе 6 данного пособия.

Обеспечение безопасности базы данных

Защита информации в информационных системах является серьезной и кропотливой задачей. На элементарном уровне ее решение сводится к обеспечению выполнения двух фундаментальных принципов: проверки полномочий пользователя — ***санкционирования доступа*** и проверки подлинности — ***аутентификации*** [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 БД. В большинстве СУБД задание и проверка полномочий определяются внутренними средствами системы. Одним из способов задания полномочий является использование операторов Grant и Revoke языка SQL. Определяются следующие уровни доступа пользователей к объектам БД: создание объекта БД; изменение; чтение; удаление; администрирование (определяет полный доступ к объекту). Сведения о полномочиях пользователя находятся в защищенной области БД. Пример окна установления прав доступа к объектам БД в СУБД MS Access представлен на рисунке 1.2.

Проверка подлинности заключается в подтверждении достоверности 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. На рисунке 1.3 представлено окно ввода имени пользователя и пароля для получения доступа к БД в СУБД MS Access.

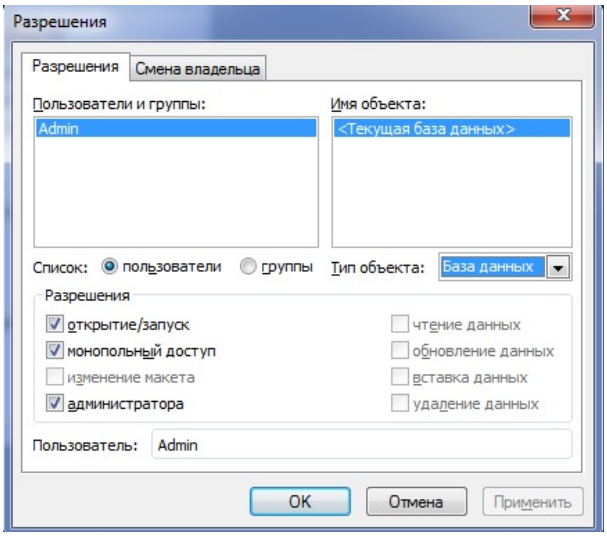


Рис. 1.2 – Окно установления прав доступа к объектам БД в СУБД MS Access

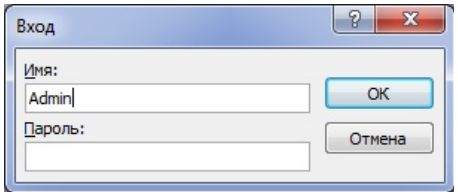


Рис. 1.3 – Окно ввода имени пользователя и пароля

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, определенному конкретно для данного пользователя, и нижним уровням.

Многоуровневая безопасность в отношении БД может строиться на основе модели, предложенной в 1975 году Дэвидом Беллом и Леонардом Лападулой — *модели Белла–Лападулы* [6]. В основе этой модели контроля и управления доступом лежит мандатная модель управления доступом. Здесь анализируются условия, при которых невозможно создание информационных потоков от субъектов с более высоким уровнем доступа к субъектам с более низким уровнем доступа. При этом объекты БД подвергаются классификации (например, особо секретно, секретно, конфиденциально, для общего пользования), а каждый пользователь причисляется к одному из уровней допуска.

Механизмы обеспечения безопасности данных постоянно совершенствуются разработчиками СУБД и являются на сегодняшний день одним из перспективных направлений в развитии информационных технологий. Организация политики безопасности в СУБД MS Access путем разделения пользователей на группы с определенными уровнями доступа представлена на рисунке 1.4.

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

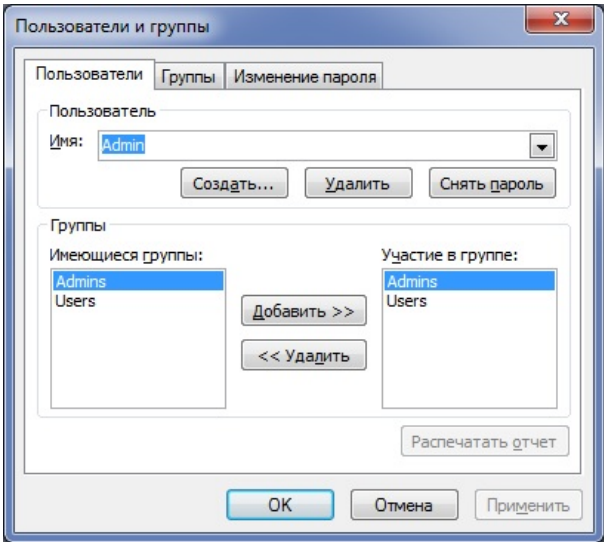


Рис. 1.4 – Разделение пользователей на группы в MS Access

Контрольные вопросы по главе 1

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. Дайте определение базы данных.
5. Дайте определение СУБД.
6. Назовите основные и дополнительные положения концепции БД.
7. Перечислите и кратко охарактеризуйте основные функции СУБД.

Глава 2

МОДЕЛИ ДАННЫХ

2.1 Архитектура представления информации в концепции баз данных

Прежде чем перейти непосредственно к описанию моделей данных, остановимся на общих принципах структуризации данных. В основе теории баз данных 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 (внешней и 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 средств проектирования и средств 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 представления информации

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 внешнего представлений можно на следующих примерах.

Концептуальное представление данных можно отобразить в виде упрощенной модели предметной области «Успеваемость студентов вуза», для чего достаточно иметь сведения о студентах и оценке, полученной студентом по конкретной дисциплине в конкретном семестре.

Для иллюстрации процесса проектирования концептуальной модели используют графическое представление информационных элементов: объектов и взаимосвязей. Стрелкой обозначим связь между двумя элементами модели, указывая, например, на соответствие каждой строке (записи) в таблице «Студент» нескольких записей в таблице «Успеваемость» (рис. 2.2).

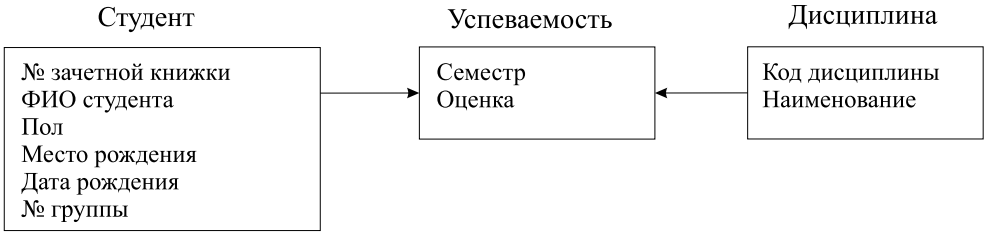


Рис. 2.2 – Концептуальное представление информации об успеваемости студентов вуза

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

При формировании *физического представления* (рис. 2.3) определяются типы данных, характерные для выбранной СУБД, создаются ключевые поля, необходимые для обеспечения уникальности данных в таблицах, а также поля, по которым будут связаны данные в разных таблицах.

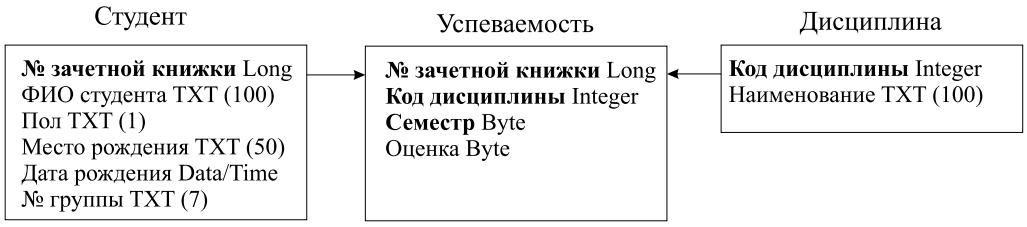
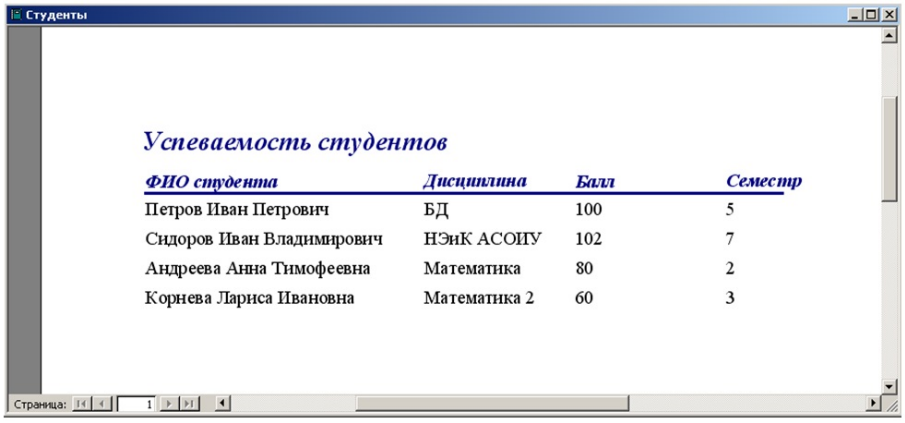


Рис. 2.3 – Физическое представление информации об успеваемости студентов вуза

Следует отметить, что физическое представление в идеологии одной СУБД может отличаться от аналогичного представления в другой, что вызвано возможными различиями в определении типов данных конкретных полей. Так, в ранних СУБД отсутствуют типы данных современных систем управления базами данных, такие как гиперссылки, OLE-объекты и др. Кроме того, существует ряд СУБД, не поддерживающих символы национальных алфавитов в именах полей и именах таблиц. По этой причине следует с особым вниманием подходить к проектированию БД, учитывая ограничения, накладываемые разработчиками СУБД.

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



№ зачетной книжки	ФИО студента	Код дисциплины	Семестр	Оценка

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

Описание концептуального и соответствующего ему физического представления (описание структуры БД) называется ***схемой БД****, хранится автономно 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 достигается 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. При этом в процессе функционирования информационных систем неизбежно возникает необходимость модификации структуры базы данных.

Внесение таких изменений в структуру базы данных, в соответствии с пользовательскими требованиями и ограничениями предметной области, называется ***эволюцией базы данных****.*

2.2 Развитие моделей данных

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

Под ***моделью данных*** *здесь будем понимать абстрактное определение объектов предметной области, операторов, а также дополнительных элементов, которые в совокупности дают возможность представления предметной области в идеологии баз данных.*

При этом объекты представляют структуру данных, а с помощью операторов моделируется поведение данных.

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

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

Иерархическая и сетевая модели относятся к классу так называемых дореляционных моделей данных, явившихся базисом для создания дореляционных СУБД.

Появление технологии организации БД послужило толчком для дальнейшего развития и совершенствования автоматизированной комплексной 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 полное представление о выбранном объекте. Отметим, что для разных объектов значения разных параметров могут относиться к одному типу данных (символьному, логическому и т. д.) и могут быть выбраны 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, называемый 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].

Здесь необходимо отметить, что существует еще один тип модели данных, не представленный выше, который в некоторых источниках [2] носит название «линейная модель данных».

Понятие структур данных было сформировано в период, предшествующий применению СУБД. Для этого периода характерно широкое использование алгоритмических языков программирования, с помощью которых пользователь мог оперировать файлами, содержащими так называемые последовательные записи. Такие файлы можно назвать файлами, содержащими простейшие ***линейные структуры*** данных. Связи между такими файлами устанавливались средствами языков программирования, что не всегда было неэффективно.

*Далее под **структурой данных** будем понимать совокупность информационных элементов и связей между ними. А собственно под **моделью данных** будем понимать соответствующий тип структуры данных и типовые операции по управлению данными в этих структурах.*

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 (или физическим представлением).

Линейную структуру данных формально можно представить в виде ***плоской таблицы***, для которой характерны следующие свойства [2]:

- элементами такой структуры являются простые данные, разделение которых на составляющие не имеет смысла;
- каждое данное (поле) имеет имя (идентификатор) и множество возможных значений, задаваемых словарем, диапазоном или правилом формирования;
- множество данных, составляющих структуру, описывает множество однотипных объектов;
- все экземпляры структуры данных (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 обеспечения уникальности экземпляров линейной структуры необходимо наличие ***первичного ключа***, представляющего 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. Если в состав первичного ключа входит более одного поля данных, такой ключ называется ***составным первичным ключом***.

На рисунке 2.5 изображена простая линейная структура, представленная в виде таблицы, содержащая однородные данные, первичным ключом в которой является «№ зачетной книжки», однако поле «№ студенческого билета» тоже однозначно определяет студента, такое поле называется ***альтернативным первичным ключом***. Заголовок линейной структуры называется схемой.

№ зачетной книжки	№ студенческого билета	ФИО студента	Пол	Место рождения	Дата рождения	№ группы
1992412-11	19957172	Карасев А. А.	М	г. Чита	27.08.95	412-1
1992432-11	19957573	Данилов О. В.	М	г. Алматы	27.08.95	432-1
1992432-12	19951775	Раевский А. И.	М	г. Бишкек	20.05.95	432-1

Рис. 2.5 – Линейная структура данных «Студенты»

Над линейными структурами возможно применение основных операций по управлению данными:

- ***вставка*** — добавление новых записей в структуру;
- ***удаление*** — удаление записей из структуры;
- ***замена*** — изменение значений данных в указанных записях;
- ***выборка*** — отбор конкретных экземпляров структуры, необходимых для дальнейшей обработки.

Для описания более сложных моделей данных (за исключением объектно-ориентированной) в качестве основной составляющей будем использовать линейные структуры данных, при этом необходимо учитывать типы взаимосвязей, возникающие между разными простыми структурами. Принято различать следующие взаимосвязи: «один-к-одному» — (1:1); «один-ко-многим» — (1:M); «многие-ко-многим» — (M:M). Связи между структурами данных являются неотъемлемой частью концептуальной модели разрабатываемой БД.

2.3 Иерархическая модель данных

Иерархическая модель данных *— это модель, в которой данные представлены в виде иерархической (древовидной) структуры, состоящей из объектов (данных) различных уровней.*

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 рисунке 2.6 представлена иерархическая структура данных (стрелка указывает направление в иерархии от старшего к подчиненному). Так, каждому экземпляру структуры СТУДЕНТЫ соответствует несколько экземпляров структуры ПЛАТА ЗА ОБУЧЕНИЕ и структуры УСПЕВАЕМОСТЬ, которой, в свою очередь, соответствует несколько экземпляров структуры КОНТРОЛЬНЫЕ ТОЧКИ.

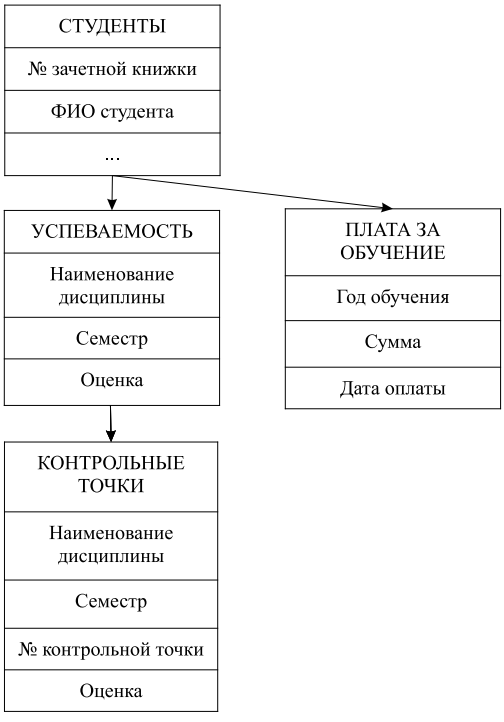


Рис. 2.6 – Иерархическая структура данных

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.

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 родительского 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****.*

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.

Применяются следующие операции по управлению данными в иерархической модели данных:

- ***включение*** — добавление экземпляров в элемент структуры (операция 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 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.

2.4 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) либо 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 (тип записи) и экземпляр элемента структуры (экземпляр записи). СТУДЕНТЫ является типом структуры, а «1112 Иванов И. И. 1987. . .» — экземпляром элемента структуры. Таким об-

разом, 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 рисунке 2.7 приведен пример простой сетевой структуры. 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:M). Элемент ДИСЦИПЛИНА также связан с элементом УСПЕВАЕМОСТЬ (1:M). 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.

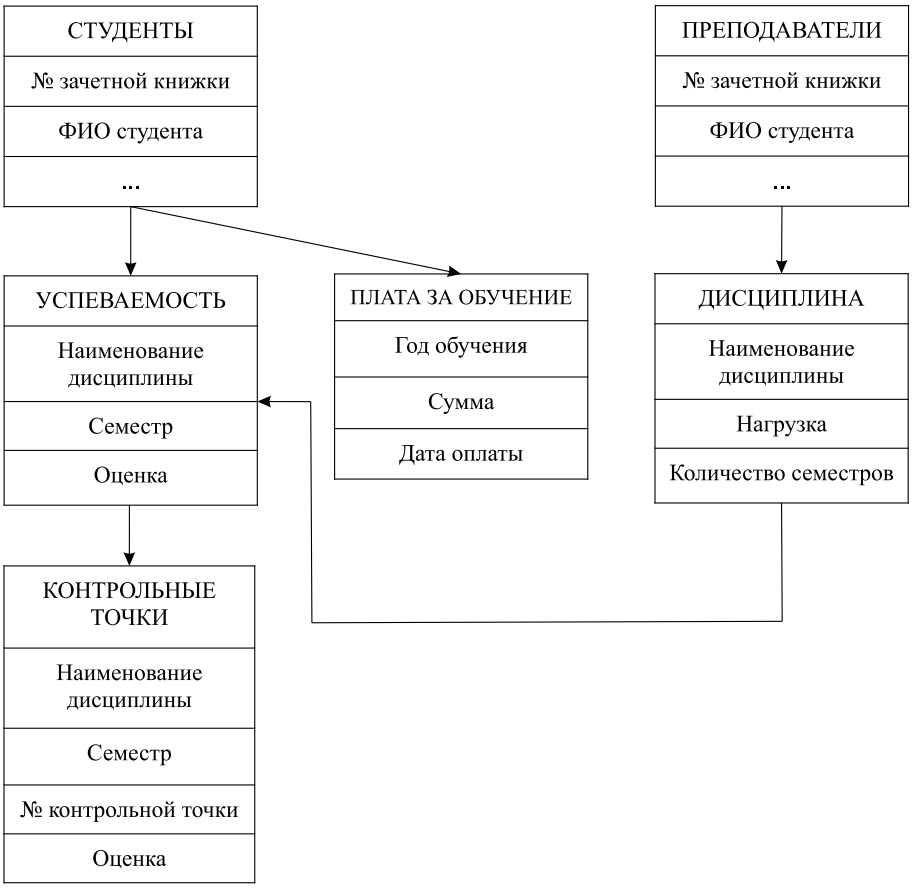


Рис. 2.7 – Сетевая структура данных

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

Контрольные вопросы по главе 2

1. Дайте определения представлений данных.
2. Дайте определение понятий схемы и подсхемы данных.
3. Назовите правила построения иерархических структур.
4. Назовите и кратко охарактеризуйте типовые операции по управлению данными в дореляционных структурах.

Глава 3

РЕЛЯЦИОННАЯ МОДЕЛЬ

3.1 Основные понятия реляционной модели

3.1.1 Общие сведения

Принципы реляционной модели данных были предложены Эдгаром Франком Коддом в 1969 году. В своей статье [7] Кодд впервые изложил основные принципы построения реляционной модели данных. При проектировании реляционной БД применяются строгие методы, построенные на нормализации отношений. Э. Ф. Кодд отмечает, что реляционная модель данных обеспечивает ряд возможностей, которые делают управление БД и их использование относительно легким, устойчивым по отношению к ошибкам и предсказуемым [8].

Термин «реляционная» говорит о том, что в основе модели лежит понятие «отношение» (англ. *relation*). Наиболее важными характеристиками реляционной модели являются следующие [8]:

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

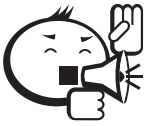
Благодаря Э. Ф. Кодду компания IBM в начале 70-х годов начала разработку нескольких коммерческих реляционных СУБД. В начале 80-х годов такие гиганты информационной индустрии, как Oracle Corporation, Ingres Corp., IBM и более мелкие организации предложили ряд систем, наглядно демонстрирующих возможность применения реляционной модели для разработки БД, хранящих информа-

цию о любой предметной области, а также возможность реализации на их основе гибких пользовательских приложений. Единственным ограничением для широкомасштабного внедрения реляционных СУБД являлась низкая производительность средств вычислительной техники [9].

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.

В реляционной модели данных принято рассматривать три основные составляющие: структурную часть, манипуляционную и целостную.

3.1.2 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 (рис. 3.1).

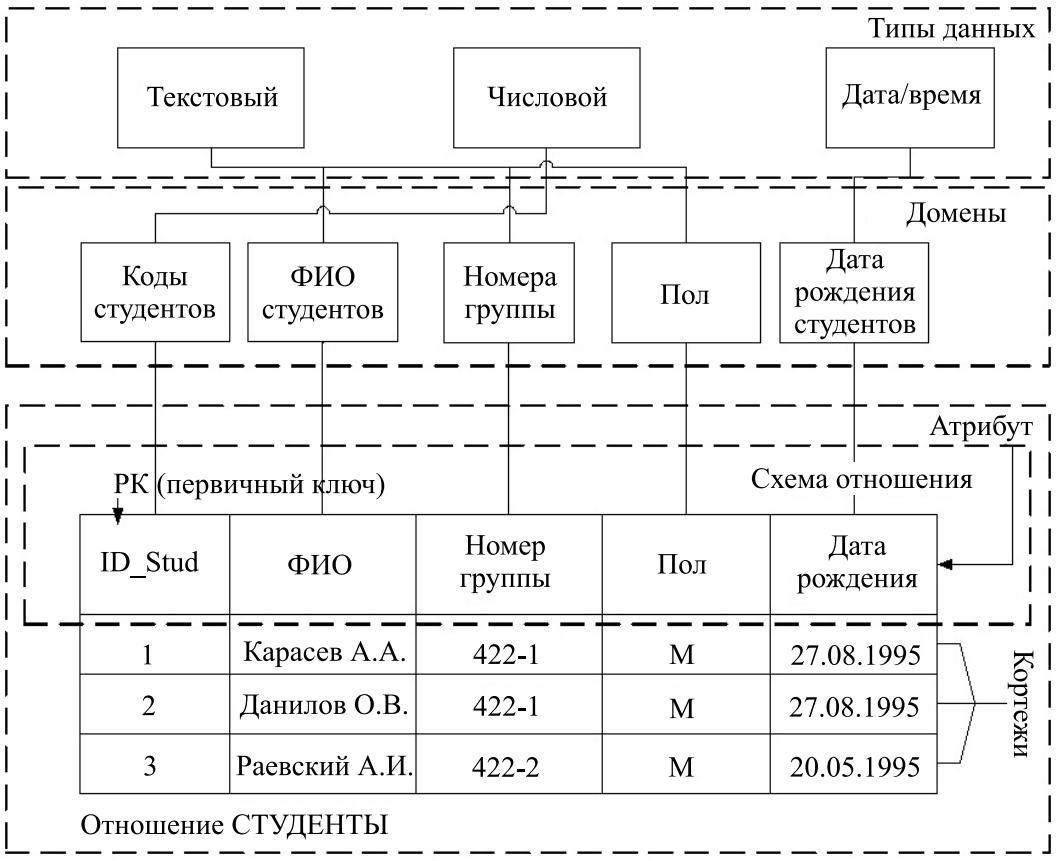


Рис. 3.1 – Отношение СТУДЕНТЫ

В реляционной модели данные представляются в виде отношений. Отношение обычно имеет простую графическую интерпретацию в виде плоской таблицы. Столбец такой таблицы называется атрибутом или полем, строка — кортежем отношения. Рассмотрим более подробно основные понятия и определения, используемые в реляционной модели данных.

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, т. е. это отношение является 5-арным. Таким образом, ***схема БД есть набор схем отношений***.

Отношение *есть множество 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.

Типам данных в реляционной модели можно сопоставить типы данных, используемых в языках программирования. Все атрибуты в отношении должны быть определенного типа. Выделяют следующие типы данных, хранящихся в реляционных БД:

- 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.

Домен *— множество допустимых значений атрибута определенного типа.*

Это понятие характерно для баз данных и 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. 3.1) значения атрибутов «Номер группы» и «Пол» относятся к текстовому типу, но не являются сравнимыми, поскольку определены на разных доменах. Если же понятие домена не поддерживается в СУБД, то можно сравнивать атрибуты, определенные на одном и том же типе данных, при этом разработчик сам определяет возможность сравнения таких атрибутов в зависимости от их смысловой нагрузки.

После описания домена можно дать обобщенное определение понятия «отношение»:

Пусть дана совокупность доменов T_1, T_2,..., T_n. Тогда ***n-арным отношением*** *R называют подмножество декартова произведения множеств T_1, T_2,..., T_n.*

Для лучшего понимания реляционной модели данных можно привести интерпретацию основных понятий (таблица 3.1).

Таблица 3.1 – Интерпретация основных понятий реляционной модели данных

Элемент реляционной модели	Форма представления в БД
Отношение	Таблица
Схема отношения	Заголовок таблицы
Кортеж	Строка таблицы (запись)
Атрибут	Поле таблицы
Имя атрибута	Имя поля таблицы
Значение атрибута	Значение поля в записи
Первичный ключ	Одно или несколько уникальных полей
Тип данных	Тип значений полей таблицы
Домен	Множество допустимых значений поля таблицы

3.2 Свойства отношений

3.2.1 Уникальность кортежей отношения

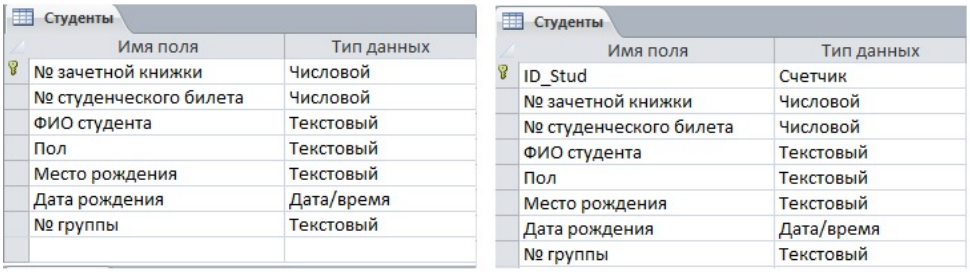
Отношениям как основной единице построения реляционных баз данных присущи определенные свойства и правила заполнения отношения сведениями об экземплярах объекта предметной области.

Главное отличие отношения от плоской таблицы заключается в том, что отношение (в классическом его понимании) не может иметь дубликатов кортежей. Поскольку отношение можно определить как множество кортежей, а каждое множество не должно включать одинаковых элементов, то отношение *не должно* иметь повторяющихся кортежей. Обеспечить это требование возможно с помощью ***первичного ключа***.

Первичный ключ — *минимально допустимый набор атрибутов, значения которых однозначно определяют кортеж отношения.*

В современных СУБД для обеспечения свойства уникальности записей в каждой таблице БД определяется первичный ключ на существующем наборе атрибутов или дополнительно вводится идентификатор записи, формируемый программно или автоматически самой СУБД (автоинкремент, счетчик и т. д.) — атрибут, значение которого является уникальным для каждой записи отношения БД. На рисунке 3.2 приведены два варианта первичных ключей — на атрибуте «№ зачетной книжки» и суррогатного первичного ключа «ID_stud».

Здесь необходимо отметить, что создание суррогатного первичного ключа целесообразно в том случае, если первичный ключ может быть определен на совокупности из трех и более атрибутов.



Студенты

Имя поля	Тип данных
№ зачетной книжки	Числовой
№ студенческого билета	Числовой
ФИО студента	Текстовый
Пол	Текстовый
Место рождения	Текстовый
Дата рождения	Дата/время
№ группы	Текстовый

Студенты

Имя поля	Тип данных
ID_Stud	Счетчик
№ зачетной книжки	Числовой
№ студенческого билета	Числовой
ФИО студента	Текстовый
Пол	Текстовый
Место рождения	Текстовый
Дата рождения	Дата/время
№ группы	Текстовый

Рис. 3.2 – Варианты определения первичных ключей в таблице Студенты

3.2.2 Отсутствие упорядоченности кортежей и атрибутов

В каком бы порядке ни хранились данные в отношении, их смысл не будет изменяться. 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 в 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.

3.2.3 Атомарность значений атрибутов, первая нормальная форма

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

*Отношение находится в **первой нормальной форме (нормализовано по 1NF)** 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. Каждое отношение в 1NF является особым случаем ненормализованного отношения, но каждое ненормализованное отношение не находится в 1NF [7]. На рисунке 3.3 приведен пример ненормализованного отношения ГРУППЫ. Можно сказать, что здесь мы имеем бинарное отношение, где значением атрибута СТУДЕНТЫ является отношение.

№ группы	Студенты				
	№ зачетной книжки	ФИО	Пол	Место рождения	Дата рождения
412-1	2014412-11	Карасев А. А.	М	г. Чита	27.08.95
412-2	2014412-02	Красников И. И.	М	г. Бийск	12.02.96
432-1	2013432-11	Данилов О. В.	М	г. Алматы	27.08.95
	2013432-12	Раевский А. И.	М	г. Бишкек	20.05.95
421-1	2014421-01	Иванкова И. С.	Ж	г. Томск	11.10.95
	2014421-02	Авдеев Н. В.	М	г. Омск	01.04.94

Рис. 3.3 – Ненормализованное отношение ГРУППЫ

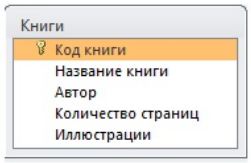
Отношение СТУДЕНТЫ (рис. 3.4) является нормализованным вариантом отношения ГРУППЫ. В этом отношении на пересечении столбца и строки находится только одно значение.

№ зачетной книжки	ФИО студента	Пол	Место рождения	Дата рождения	№ группы
2014412-11	Карасев А. А.	М	г. Чита	27.08.95	412-1
2014412-02	Красников И. И.	М	г. Бийск	12.02.96	412-2
2013432-11	Данилов О. В.	М	г. Алматы	27.08.95	432-1
2013432-12	Раевский А. И.	М	г. Бишкек	20.05.95	432-1
2014421-01	Иванкова И. С.	Ж	г. Томск	11.10.95	421-1
2014421-02	Авдеев Н. В.	М	г. Омск	01.04.94	421-1

Рис. 3.4 – Отношение СТУДЕНТЫ, находящееся в 1NF

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 предметной области для обеспечения возможности манипулирования информацией, представленной в этом поле, что влечет необходимость дальнейшей нормализации; либо эта информация принимается как сопроводительная и имеет статус примечания.

Однако в ряде случаев крайне необходимо избавиться от такой «неявной ненормализованности» отношений. Рассмотрим следующий пример: пусть в БД хранится информация о книгах (таблица «Книги»). В поле «Автор» хранится информация об авторах книги (рис. 3.5).



Книги

Код книги	Название книги	Автор	Количество страниц	Иллюстрации
1	Война и мир	Л.Н. Толстой	801	☑
2	Пикник на обочине	А.Н. Стругацкий, Б.Н. Стругацкий	205	☑
3	12 стульев	И. Ильф, Е. Петров	312	☐

Рис. 3.5 – Фрагмент таблицы «Книги»

Легко заметить, что в том случае если одна книга написана несколькими авторами, это поле будет содержать множественные значения, при этом вряд ли такое

поле может «иметь статус примечания». При работе с такой таблицей могут возникнуть трудности как при её заполнении, так и при выполнении различного рода операций по выборке данных, например при поиске и подсчете книг определенного автора.

Для выполнения нормализации в данном случае необходимо провести ряд преобразований. Во-первых, необходимо создать справочник-классификатор «Авторы», где будет храниться информация обо всех авторах. Во-вторых, необходимо создать промежуточную таблицу «Авторы_книг», которая позволит связать таблицы «Книги» и «Авторы», обеспечив возможность введения информации о том, что «один автор может написать несколько книг, а одна книга может быть написана несколькими авторами». В-третьих, осталось удалить поле «Автор» из таблицы «Книги», как содержащее избыточную информацию (рис. 3.6, 3.7).

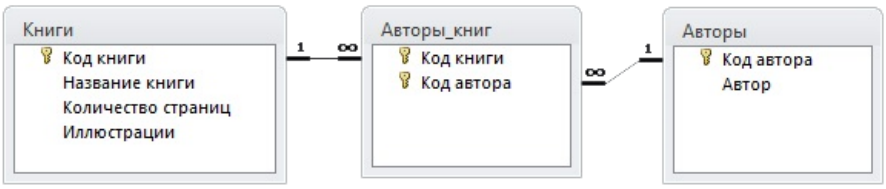


Рис. 3.6 – Схема данных БД после нормализации таблицы «Книги»

Авторы

Код автора	Автор
1	Л.Н. Толстой
2	А.Н. Стругацкий
3	Б.Н. Стругацкий
4	И. Ильф
5	Е. Петров

Авторы_книг

Код книги	Код автора
1	1
2	2
2	3
3	4
3	5

Книги

Код книги	Название книги	Количество страниц	Иллюстрации
1	Война и мир	801	☑
2	Пикник на обочине	205	☑
3	12 стульев	312	☐

Рис. 3.7 – Фрагменты таблиц «Книги», «Авторы_книг» и «Авторы»

Кроме этого, как отмечалось выше, в современных реляционных СУБД допускается хранение в полях таблиц сложных объектов (например, полей 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.

3.2.4 Состав реляционной модели данных

В предыдущих главах, говоря о иерархической и сетевой моделях данных, было указано, что эти модели состоят из двух частей — структурной и манипуляционной. Что касается реляционной модели данных, то публикаций на эту тему в настоящее время достаточно много, однако фундаментальные понятия, термины

и основы построения реляционной модели остаются неизменными на протяжении десятилетий. Основы проектирования реляционных моделей подробно изложены в [7–10]. Так, Кристофер 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***.

Что касается ***целостной части*** реляционной модели данных, то для нее фиксируются четыре базовых требования, характерные для реляционных СУБД:

- первое требование, согласно которому отношение должно обладать первичным ключом для обеспечения уникальности записей, называется требованием целостности сущностей;
- второе — требование целостности по ссылкам или ссылочная целостность;
- третье — требование целостности доменов;
- четвертое — требование целостности, определяемой пользователем.

В общем смысле целостность базы данных можно рассматривать как соответствие информации, хранящейся в базе данных, её структуре, внутренней логике, всем правилам её формирования и требованиям предметной области. Каждое правило, которое налагает некоторое требование (ограничение) на возможное состояние базы данных, называется ограничением целостности (*integrity constraint*).

Одной из задач специалиста, занятого анализом предметной области (аналитика), и проектировщика базы данных — выявить как можно более полно все имеющиеся ограничения целостности и задать их в базе данных. Однако при этом необходимо учитывать, что целостность БД, определенная в процессе ее проектирования, не гарантирует достоверности содержащейся в ней информации (за это может отвечать оператор — пользователь информационной системы, вводящий информацию в БД). При этом обеспечивается, по крайней мере, корректность вводимой информации — не допускается ввод заведомо ложных значений (например, невозможность ввода в одно поле значений, определенных на домене, соответствующем другому полю).

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.

3.3 Целостная часть реляционной модели данных

3.3.1 Целостность сущности

Что касается целостности сущности, то это понятие напрямую связано со свойством уникальности кортежей отношения (см. раздел 3.2.1) — каждый кортеж отношения должен отличатся от любого другого кортежа этого отношения, т. е. любое отношение должно обладать первичным ключом.

Здесь необходимо отметить главное требование к формированию значения первичного ключа — атрибуты, входящие в состав первичного ключа, не могут принимать несуществующие (null)-значения.

Таким образом, при добавлении новых кортежей в отношение проверяется значение формируемых (или вводимых) их первичных ключей, а при изменении значения первичного ключа необходимо проверять его уникальность.

3.3.2 Ссылочная целостность

Целостность по ссылкам, или ссылочная целостность, может рассматриваться как набор задаваемых на стадии проектирования БД правил, обеспечивающих логическую согласованность первичных и внешних ключей при вставке, обновлении и удалении кортежей отношений.

При описании объектов предметной области 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 из раздела 3.2. В результате проектирования БД «ВУЗ» схема данных содержит три отношения (таблицы) «Студент», «Дисциплины» и «Успеваемость» со схемами, представленными на рисунке 3.8.

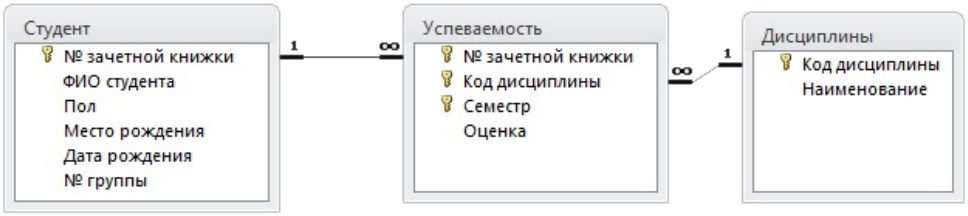


Рис. 3.8 – Фрагмент схемы данных БД «ВУЗ»

Атрибуты «№ зачетной книжки» и «Код дисциплины» появляются в отношении «Успеваемость» для обеспечения возможности восстановить полную информацию об успеваемости студентов вуза. Значение атрибута «№ зачетной книжки» отношения «Успеваемость» должно соответствовать значению атрибута «№ зачетной книжки» в каком-либо кортеже отношения «Студент». Значение атрибута «Код дисциплины» отношения «Успеваемость» должно соответствовать значению атрибута «Код дисциплины» в каком-либо кортеже отношения «Дисциплины».

Атрибуты «№ зачетной книжки» и «Код дисциплины» в отношении «Успеваемость» называются ***внешними ключами***.

Внешний ключ — *это атрибут (или совокупность атрибутов), значения которого однозначно характеризуют сущности, представленные кортежами другого отношения, т. е. соответствуют значению его первичного ключа.*

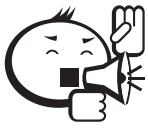
Таким образом, отношение, в котором на каком-либо атрибуте (может быть составным) определен внешний ключ, ссылается на отношение, в котором соответствующий атрибут является первичным ключом. Говорят также, что отношения связаны по некоторому ключу.

Требование целостности по ссылкам (требование внешнего ключа) состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, должен найтись кортеж с таким же значением первичного ключа в отношении, на которое ведет ссылка, либо значение внешнего ключа должно быть неопределенным, т. е. ни на что не указывать [1].

Для нашего примера это означает, что если в успеваемости студента указан код дисциплины, то эта дисциплина 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, поддерживающих 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.

Можно говорить, что любая база данных обладает свойством ссылочной целостности, когда для любой пары связанных внешним ключом отношений в ней выполняется условие ссылочной целостности.

3.3.3 Целостность доменов

Для реляционных СУБД, поддерживающих понятие доменов или возможность задания ограничений на значение какого-либо поля, существует понятие целостности доменов.

*Обеспечение **механизма целостности доменов** гарантирует, что все значения некоторого атрибута принадлежат множеству допустимых значений.*

На рисунке 3.9 представлен механизм реализации целостности доменов в БД СУБД MS Access. Хотя в явном виде понятие доменов в этой СУБД не поддерживается, разработчик БД имеет возможность ограничить диапазон значений определенного поля. В нашем примере задается ограничение на набор значений поля «Пол» таблицы «Студент».

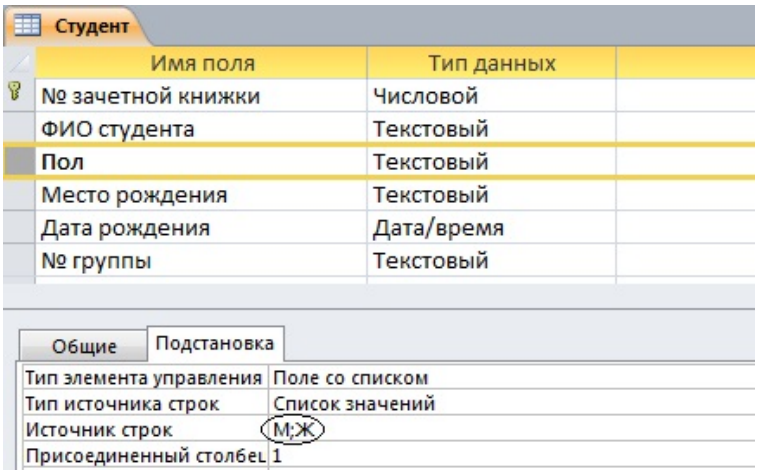


Рис. 3.9 – Задание ограничения для формирования поля «Пол»

На рисунке 3.10 представлено окно редактирования таблицы «Студенты» с выбором определенного значения для поля «Пол».

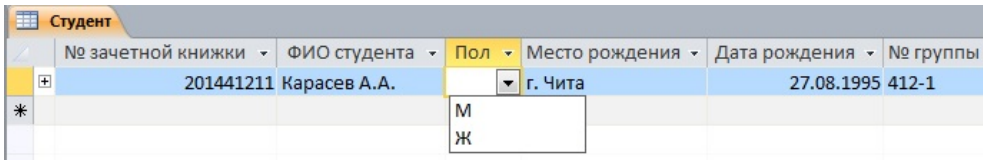
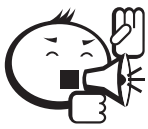


Рис. 3.10 – Заполнение поля «Пол»

Таким образом, реализация механизма целостности доменов осуществляется с помощью предварительного задания характеристик домена в описательной части БД.

3.3.4 Целостность, определяемая пользователем

Целостность, определяемая пользователем, расширяет понятия целостности доменов и ссылочной целостности. Так, пользователь и/или проектировщик БД может задать дополнительные ограничения на набор возможных значений поля или совокупности полей.



При разработке БД выделяется особый класс таблиц — справочники-классификаторы, это очень удобный и очень полезный способ задания перечисляемых полей.

Изменим БД (рис. 3.8), добавив в нее справочник-классификатор «Город», для указания места рождения студентов (рис. 3.11).

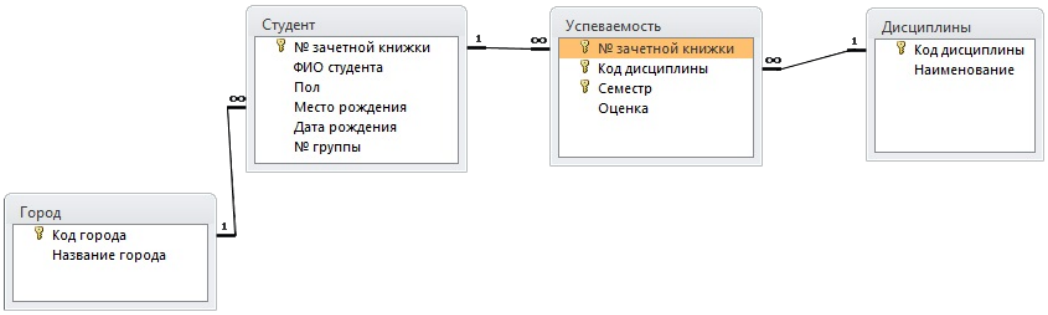


Рис. 3.11 – Измененная схема БД «ВУЗ»

Чтобы корректно связать этот справочник городов с отношением «Студенты», нам придется изменить тип данных атрибута «Место рождения» с текстового на числовой, а также указать источник строк для формирования атрибута «Место рождения» (рис. 3.12).

При работе с таблицей «Студент» необходимо, чтобы были внесены значения справочника-классификатора «Город», чтобы пользователю было удобно формировать значения атрибута «Место рождения» простым выбором значений из выпадающего списка (нижняя часть рисунка 3.12).

Использование справочников-классификаторов позволяет добиться не только удобства при заполнении полей, но и обеспечивает защиту от возникновения так называемых «висящих» ссылок при обеспечении ссылочной целостности.

Кроме явного указания ограничений на значения атрибута путем задания допустимого набора значений при проектировании БД (фиксированные значения домена) или при работе с БД (формирование пользователем справочников-классификаторов), возможно также задание дополнительных ограничений. Например, задание ограничения на формирование совокупности атрибутов «Серия» и «№ паспорта» — так, в отдельности значения этих атрибутов могут повторяться, но их совокупное значение должно быть всегда уникальным.

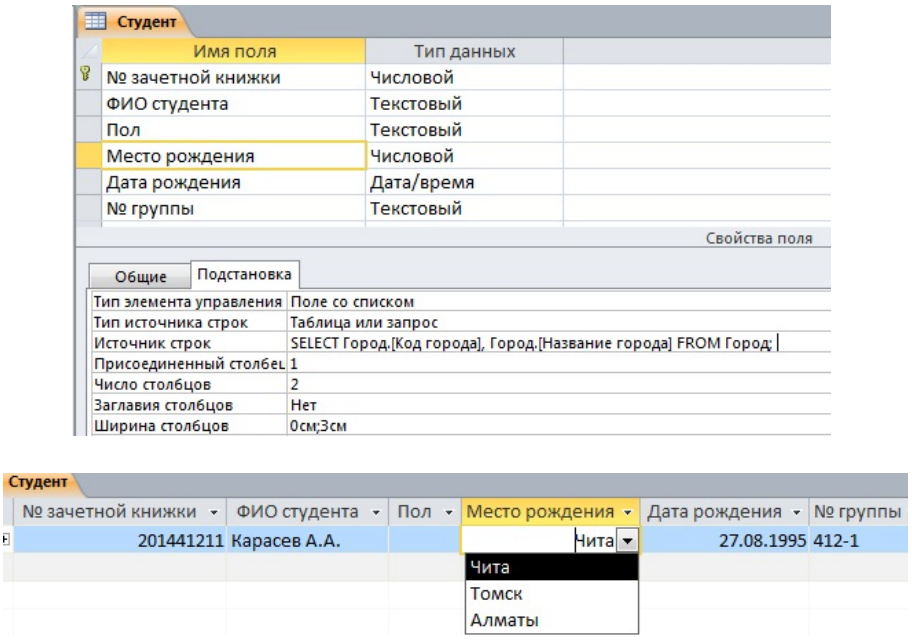


Рис. 3.12 – Указание источника строк и заполнение атрибута «Место рождения»

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.

3.4 Технология манипулирования данными в реляционной модели

3.4.1 Основные принципы технологии манипулирования реляционными данными

Согласно К. Дейту для манипуляционной составляющей реляционной модели данных определяются два базовых механизма манипулирования реляционными данными — реляционная алгебра (основана на теории множеств) и реляционное исчисление (основано на исчислении доменов и исчислении 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.

3.4.2 Операции реляционной алгебры

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 выше, есть множества, то для них справедливо применение теоретико-множественных операций, дополненных некоторыми специальными операциями, специфичными для баз данных.

Набор основных операций реляционной алгебры включает восемь операций, среди которых выделяют:

1) традиционные теоретико-множественные операции:
   - *прямое произведение отношений*;
   - *объединение отношений*;
   - *пересечение отношений*;
   - *взятие разности отношений*;
2) четыре специальные реляционные операции:
   - *ограничение отношения*;
   - *проекция отношения*;
   - *соединение отношений*;
   - *деление отношений*.

Этот набор дополняет *операция присваивания*, сохраняющая в базе данных результаты вычисления, и *операция переименования атрибутов*, дающая возможность корректировать схему результирующего отношения.

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 подробно рассмотреть операции реляционной алгебры, дадим следующие общие определения.

*Результатом выполнения **прямого 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 с множеством значений атрибутов второго операнда.*

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

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.

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.

В целях 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.

Операции объединения, пересечения и разности

Для парных теоретико-множественных операций реляционной алгебры необходимо выполнение некоторых условий. Так, для операций объединения, пересечения и разности — условия 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 корректно выполняется в том и только 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 операции объединения представлен на рисунке 3.13.

*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.*

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

Отношение «Студенты 1-го курса»

№ студента	ФИО студента	Пол	Место рождения	Дата рождения	№ группы
1	Карасев А. А.	М	г. Чита	27.08.95	412-1
2	Данилов О. В.	М	г. Алматы	27.08.95	432-1
3	Раевский А. И.	М	г. Бишкек	20.05.95	432-1

Отношение «Абитуриенты»

№ абитуриента	ФИО абитуриента	Пол	Место рождения	Дата рождения	№ группы
1001	Иванкова И. С.	Ж	г. Томск	11.10.95	421-1
1002	Авдеев Н. В.	М	г. Омск	01.04.94	421-1
1003	Красников И. И.	М	г. Бийск	12.02.97	412-2

Отношение «Студенты»

№ студента	ФИО студента	Пол	Место рождения	Дата рождения	№ группы
1	Карасев А. А.	М	г. Чита	27.08.95	412-1
2	Данилов О. В.	М	г. Алматы	27.08.95	432-1
3	Раевский А. И.	М	г. Бишкек	20.05.95	432-1
1001	Иванкова И. С.	Ж	г. Томск	11.10.95	421-1
1002	Авдеев Н. В.	М	г. Омск	01.04.94	421-1
1003	Красников И. И.	М	г. Бийск	12.02.97	412-2

Рис. 3.13 – Пример выполнения операции объединения

Прямое 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. 3.14). 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.

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.

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. 3.14 – Пример операции прямого произведения

Отметим, что вышеперечисленные операции являются ассоциативными. Обозначим через OP любую из четырех операций, тогда (R_1 OP R_2) OP R_3 = R_1(R_2 OP OP R_3) и 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.

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 осмыслена операция сравнения comp-op; либо вид (a comp-op const), где a — имя атрибута ограничиваемого отношения, a 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 кортежи отношения-операнда, для которых выполняется условие ограничения. Результат операции ограничения представляется в виде горизонтальной «вырезки» кортежей отношения-операнда. На рисунке 3.15 приведено отношение «Студенты» (см. рис. 3.13), к которому применена операция ограничения с условием № группы равен «421-1».

Рассмотрим механизм применения операции ограничения. 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).

Отношение «Студенты»

№ студента	ФИО студента	Пол	Место рождения	Дата рождения	№ группы
1001	Иванкова И. С.	Ж	г. Томск	11.10.95	421-1
1002	Авдеев Н. В.	М	г. Омск	01.04.94	421-1

Рис. 3.15 – Результат выполнения операции ограничения

Итак, допускается использование операций ограничения, где 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,…,r_n является отношение с заголовком, определяемым множеством атрибутов r_1, r_2,…,r_n, и телом, состоящим из кортежей вида $<r_1 : v_1, r_2 : v_2,\ldots,r_n : v_n>$ таких, что в отношении R имеется кортеж, атрибут r_1 которого имеет значение v_1, атрибут r_2 имеет значение v_2,…, атрибут r_n имеет значение v_n.*

Тем самым при выполнении операции проекции выделяется «вертикальная» вырезка отношения-операнда [1]. Заметим, что при появлении дублирующих кортежей необходимо исключить таковые из результирующего набора. Так, результатом выполнения операции взятия проекции отношения «Студенты» (см. рис. 3.13) на атрибуты «№ студента», «ФИО студента», «№ группы» является отношение «Группа студентов» (рис. 3.16).

Отношение «Группа студентов»

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

Рис. 3.16 – Результат операции взятия проекции

Операция соединения отношения

Результатом выполнения ***операции соединения отношения*** R_1 *по атрибуту* X *с отношением* R_2 *по атрибуту* Y *является отношение* R*, множество всех кортежей которого является конкатенацией таких кортежей* $a \in R_1$ *и кортежей* $b \in R_2$*, 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 рисунке 3.17 приведен пример использования операции соединения отношений $R_1(A_1, A_2, A_3)$ и $R_2(A_3, A_4)$ по совпадению значений атрибутов A_3 в обоих отношениях-операндах — т. е. производится сопоставление преподавателей с их должностями.

R1 («Преподаватели»)

A_1 (Код преподавателя)	A_2 (ФИО преподавателя)	A_3 (Код должности)
1	Смирнов А. Н.	1
3	Смольников В. В.	3
2	Гринько О. А.	2

R2 («Должности»)

A_3 (Код должности)	A_4 (Название должности)
1	Доцент
2	Профессор
3	Лаборант

R («Должности преподавателей»)

A_1 (Код преподавателя)	A_2 (ФИО преподавателя)	A_3 (Код должности)	A_4 (Название должности)
1	Смирнов А. Н.	1	Доцент
2	Гринько О. А.	2	Профессор
3	Смольников В. В.	3	Лаборант

Рис. 3.17 – Пример операции соединения

Операция деления отношений

Кроме определения операции деления, приведенного в начале данного раздела, справедливо следующее определение.

Пусть даны два отношения $R(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$ *— делимое и* $S(B_1, B_2, \ldots, B_m)$ *— делитель, где атрибуты* $B_1, B_2, \ldots, B_m$ *общие для обоих отношений, тогда результатом* ***деления отношений*** R/S *будет отношение со схемой* $(A_1, A_2, \ldots, A_n)$ *и телом, со-*

держащим множество кортежей $(a_1, a_2, \ldots, a_n)$ *таких, что для всех кортежей* $(b_1, b_2, \ldots, b_m) \in S$ *в отношении* R *должен существовать кортеж* $(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m)$.

. . . . . .

Рассмотрим пример: пусть в БД имеются два отношения «ВУЗ» и «Направления подготовки», необходимо найти вузы, в которых осуществляется обучение по всем направлениям подготовки (рис. 3.18).

«ВУЗ» (R)

Наименование вуза	Наименование направления
ТУСУР	Государственное и муниципальное управление
ТУСУР	Программная инженерия
ТГУ	Бизнес-информатика
ТГУ	Государственное и муниципальное управление
ТПУ	Программная инженерия
ТУСУР	Бизнес-информатика

/ «Направление подготовки» (S)

Наименование направления
Государственное и муниципальное управление
Программная инженерия
Бизнес-информатика

Результат

Наименование вуза
ТУСУР
ТУСУР
ТУСУР

Рис. 3.18 – Пример выполнения операции деления

3.4.3 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].

Реляционное исчисление основано на исчислении кортежей, где областью определения переменных являются тела отношений БД, т. е. допустимым значением каждой переменной является кортеж тела отношения, и на исчислении доменов, где областью определения переменных являются домены, на которых определены атрибуты отношения, т. е. допустимым значением каждой переменной является значение домена.

Исчисление кортежей было введено Коддом в 1971 году — это нотация для определения результата запроса через описание его свойств. В исчислении кортежей задача состоит в нахождении таких кортежей, для которых предикат является истинным.

Пусть имеется БД, в состав которой входят два отношения «Студенты» с заголовком (Stud_id, № зачетной книжки, ФИО студента, Место рождения, Дата рождения, № группы) и отношение «Успеваемость» с заголовком (Stud_id, Наименование дисциплины, Семестр, Оценка). Необходимо получить 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 = Успеваемость.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 студента», «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, для которых существует кортеж в отношении «Успеваемость» с таким же значением «Stud_id», что и в отношении «Студенты», при этом выполняется условие назначения атрибутов Семестр = 2 и Оценка = 2.

Здесь мы указали характеристики результирующего набора кортежей, не указав способ его формирования. Для выполнения подобных запросов в СУБД предусмотрены процедуры выбора необходимых операций по обработке данных, а также их последовательности. 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].

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: $\exists$ var (F); $\forall$ var (F).

Переменные, входящие в правильно построенную формулу, могут быть связанными 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 ∃ 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 увидеть, 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].

Пример 3.1

В следующем примере мы имеем несколько разнотипных вхождений переменных «Студенты» и «Успеваемость».

∃ 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.

Как было отмечено выше, в исчислении доменов областью определения переменных являются не тела отношений, а домены. Можно говорить о доменных переменных, таких как «ФИО» (значения — допустимые ФИО студентов) или «N_Group» (значения — допустимые номера групп) и т. д.

Формальным отличием исчисления доменов от исчисления кортежей является наличие дополнительного набора предикатов, позволяющих выражать условия членства.

В заключении раздела отметим, что на основе операций реляционной алгебры и реляционного исчисления строятся языки манипулирования данными, наиболее распространенным из которых является язык SQL, наиболее гибко сочетающий в себе операции реляционной алгебры и реляционное исчисление.

Контрольные вопросы по главе 3

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. Перечислите и кратко охарактеризуйте операции реляционной алгебры.
6. Приведите примеры выполнения операций реляционной алгебры.
7. Поясните смысл применения реляционного исчисления.

Глава 4

ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

4.1 Нормализация отношений

4.1.1 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. Можно выделить две стадии проектирования базы данных — логическое и физическое проектирование.

Логическое проектирование БД *ставит своей целью представление реальной предметной области в абстрактных моделях таким образом, 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 предусмотрено 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.

Одной из направляющих при проектировании реляционных БД является технология нормализации отношений, направленная на обеспечение безызбыточного и бесконфликтного хранения информации в отношениях БД. При использовании 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 NF соответствует свой набор ограничений, тогда справедливо утверждение, что ***отношение находится в какой-либо нормальной форме, если удовлетворяет требуемому ей набору ограничений***.

Так, ограничением первой нормальной формы (1NF), как мы отмечали в третьем разделе, является условие атомарности атрибутов отношения. И поскольку в основе реляционной модели лежит отношение, находящееся в 1NF, в дальнейшем будем подразумевать, что все рассматриваемые нами отношения уже находятся в 1NF или, как говорят, нормализованы по первой нормальной форме. Процесс нормализации позволяет разработчику БД глубже понять смысл атрибутов отношений 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: $R.X \to R.Y$ или $R(X \to Y)$.

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 $X \rightarrow Y$ *называется* ***транзитивной****, если имеется такой атрибут Z, что существуют функциональные зависимости* $X \rightarrow Z$ *и* $Z \rightarrow Y$*, и при этом отсутствует функциональная зависимость* $Z \rightarrow X$.

В отношении R атрибуты ***взаимно независимы****, если ни один из этих атрибутов функционально не зависит от других.*

4.1.2 Вторая нормальная форма

Рассмотрим процесс нормализации отношения по 2NF, т. е. приведем отношение (рис. 4.1), находящееся в первой нормальной форме, ко второй нормальной форме.

Отношение «Успеваемость»

№ зачетной книжки	ФИО студента	Место рождения	Дата рождения	**Курс**	Средний балл
2014412-11	Карасев А. А.	г. Чита	27.08.95	1	4,5
2014412-11	Карасев А. А.	г. Чита	27.08.95	2	4,1
2014412-11	Карасев А. А.	г. Чита	27.08.95	3	5
2014412-11	Карасев А. А.	г. Чита	27.08.95	4	4,8
2014432-11	Данилов О. В.	г. Алматы	27.08.95	1	4,6
2014432-11	Данилов О. В.	г. Алматы	27.08.95	2	4,4
2014432-11	Данилов О. В.	г. Алматы	27.08.95	3	4,2
2014432-11	Данилов О. В.	г. Алматы	27.08.95	4	4,3
2014432-12	Раевский А. И.	г. Бишкек	20.05.95	1	3,8
2014432-12	Раевский А. И.	г. Бишкек	20.05.95	4	5

Рис. 4.1 – 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 совокупности 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».

При работе с таким ненормализованным отношением невозможно обеспечить корректную работу по выполнению 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 2NF.

Отношение R находится во ***второй нормальной форме (2NF)*** *тогда и только тогда, когда отношение находится в первой нормальной форме и каждый его неключевой 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 2NF 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. 4.2). Первичным ключом отношения «Студенты» является атрибут «№ зачетной книжки».

«Студенты»

№ зачетной книжки	ФИО студента	Место рождения	Дата рождения
2014412-11	Карасев А. А.	г. Чита	27.08.95
2014432-11	Данилов О. В.	г. Алматы	27.08.95
2014432-12	Раевский А. И.	г. Бишкек	20.05.95

«Успеваемость студентов»

№ зачетной книжки	**Курс**	Средний балл
2014412-11	1	4,5
2014412-11	2	4,1
2014412-11	3	5
2014412-11	4	4,8
2014432-11	1	4,6
2014432-11	2	4,4
2014432-11	3	4,2
2014432-11	4	4,3
2014432-12	1	3,8
2014432-12	4	5

Рис. 4.2 – Результат декомпозиции отношения «Успеваемость»

Можно выявить следующие функциональные зависимости:

- «№ зачетной книжки» → «ФИО студента»;
- «№ зачетной книжки» → «Место рождения»;
- «№ зачетной книжки» → «Дата рождения».

Первичным ключом отношения «Успеваемость студентов» является совокупность атрибутов «№ зачетной книжки», «Курс».

В этом отношении существует одна функциональная зависимость: «№ зачетной книжки», «Курс» → «Средний балл».

Каждое из этих двух отношений находится в 2NF, 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 2NF примет следующий вид.

Отношение R находится во ***второй нормальной форме (2NF)*** *в том и только в том случае, когда оно находится в 1NF и каждый его неключевой атрибут полностью зависит от каждого потенциального ключа этого отношения.*

4.1.3 Третья нормальная форма

Добавим в отношение «Студенты» два атрибута: «№ группы» и «ФИО куратора» (рис. 4.3). Первичным ключом отношения «Студенты» является атрибут «№ зачетной книжки». Отношение находится во второй нормальной форме, поскольку отсутствуют зависимости атрибутов от части первичного ключа.

№ зачетной книжки	ФИО студента	Дата рождения	Место рождения	№ группы	ФИО куратора
2014412-11	Карасев А. А.	27.08.95	г. Чита	412-1	Самойлов С. С.
2014432-11	Данилов О. В.	27.08.95	г. Алматы	432-1	Авдеев Р. М.
2014432-12	Раевский А. И.	20.05.95	г. Бишкек	432-1	Авдеев Р. М.
2014432-22	Глазов О. А.	04.07.95	г. Киров	432-1	Авдеев Р. М.

Рис. 4.3 – Отношение «Студенты», 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, в свою очередь, существуют некоторые явные аномалии включения и удаления. Так, в результате выполнения операции добавления мы не сможем занести в отношение 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 (3NF)*** *в том и только в том случае, если находится в 2NF 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 3NF (нормализация по 3NF) также заключается в декомпозиции этого отношения. Можно произвести декомпозицию отношения «Студенты» в два отношения «Справочник студентов» и «Кураторы групп». Результат нормализации представлен на (рис. 4.4).

«Справочник студентов»

№ зачетной книжки	ФИО студента	Дата рождения	Место рождения	№ Группы
2014412-11	Карасев А. А.	27.08.95	г. Чита	412-1
2014432-11	Данилов О. В.	27.08.95	г. Алматы	432-1
2014432-12	Раевский А. И.	20.05.95	г. Бишкек	432-1
2014432-22	Глазов О. А.	04.07.95	г. Киров	432-1

«Кураторы групп»

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

Рис. 4.4 – Результат декомпозиции отношения «Студенты»

Первичным ключом отношения «Кураторы групп» является атрибут «№ группы». Для этого отношения характерна одна функциональная зависимость: «№ группы» → «ФИО куратора».

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

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».

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.

4.1.4 Нормальная форма Бойса—Кодда

Нормальная форма Бойса—Кодда (BCNF) является расширением третьей нормальной формы, вводит дополнительное ограничение по сравнению с 3NF и является промежуточным звеном между 3NF и 4NF. Нормальную форму Бойса—Кодда принято также называть *усиленной третьей 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 атрибута «ФИО студента» (рис. 4.5).

«Успеваемость по дисциплинам»

№ зачетной книжки	**ФИО студента**	**Код дисциплины**	Рейтинг
2014412-11	Карасев А. А.	1	100
2014412-11	Карасев А. А.	2	90
2014412-11	Карасев А. А.	3	98
2014432-22	Глазов О. А	1	111
2014432-22	Глазов О. А	4	122
2014432-12	Раевский А. И.	1	101
2014432-12	Раевский А. И.	2	120
2014432-12	Раевский А. И.	3	95

Рис. 4.5 – 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 3NF 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. 4.6).

«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. 4.6 – Отношения, находящиеся в BCNF

Тогда возможными ключами отношения «Студенты» являются «№ 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».

4.1.5 Четвертая нормальная форма

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

В отношении $R(A, B, C)$ существует ***многозначная зависимость (MVD)*** *$A \rightarrow\rightarrow B$ 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; атрибуты B, C многозначно зависят от A ($A \rightarrow\rightarrow B|C$) 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 отношения «Изучаемые дисциплины» (рис. 4.7). 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. 4.7 – 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 явных функциональных зависимостей, то декомпозиция отношения не может быть выполнена на основе функциональных зависимостей. Однако заметно, что какая-то взаимосвязь между атрибутами существует. Эта взаимосвязь и есть многозначная зависимость.

В отношении «Изучаемые дисциплины» существуют две многозначные зависимости:

- «Предмет» $\rightarrow\!\!\rightarrow$ «Код специальности» — зависимость множества значений атрибута «Код специальности» от множества значений атрибута «Предмет»;
- «Предмет» $\rightarrow\!\!\rightarrow$ «Литература» — зависимость множества значений атрибута «Литература» от множества значений атрибута «Предмет».

В общем случае в отношении $R(A,\ B,\ C)$ существует многозначная зависимость $A \rightarrow\!\!\rightarrow B$ в том и только в том случае, когда существует многозначная зависимость $A \rightarrow\!\!\rightarrow C$ [1].

Многозначная зависимость $A \rightarrow\!\!\rightarrow B|C$ называется нетривиальной, если не существует функциональных зависимостей $A \rightarrow B$ и $A \rightarrow C$.

В нашем примере мы имеем дело именно с нетривиальной многозначной зависимостью.

Нормализация отношений по 4NF 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 многозначная зависимость $A \rightarrow\!\!\rightarrow B|C$.

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 **4NF** в том и только в том случае, если в случае существования многозначной зависимости $A \rightarrow\!\!\rightarrow B$ 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-специальность» и «Предметы-литература»:

- «Предметы-Специальность» («Предмет», «Код специальности»);
- «Предметы-Литература» («Предмет», «Литература»).

Оба эти отношения находятся в 4NF 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 одном 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.

4.1.6 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 «Специальность студента» (рис. 4.8).

«Специальность студента»

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. 4.8 – Пример отношения, не находящегося в 5NF

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 4NF. Однако в нем могут существовать аномалии, которые нельзя устранить путем декомпозиции этого отношения в два. Всевозможные проекции отношения, включающие по два атрибута, представлены на рисунке 4.9.

Из этого примера нетрудно заметить, что отношение R не восстанавливается ни по одному из попарных соединений R_1 и 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. 4.10).

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.

Проекция R_1=R[A, B]

Код студента	№ группы
1	422-1
1	472-1-В
2	422-1

Проекция R_2=R[A ,C]

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

Проекция R_3=R[B, C]

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

Рис. 4.9 – Проекции отношения «Специальность студентов»

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. 4.10 – «Восстановленное» отношение «Специальность студентов»

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)*** *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 5NF.

*Отношение находится в **5NF** тогда и только тогда, когда любая зависимость по соединению в нем определяется только его возможными ключами.*

*Отношение R находится в **пятой нормальной форме (5NF)** 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 нашего отношения в три новых отношения, и выглядеть она будет 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 5NF.

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

4.1.7 Денормалилизация отношений

В некоторых случаях разработчики баз данных специально вносят различного рода избыточности в БД, добиваясь тем самым повышения быстродействия при выполнении запросов с ущербом для нормализации. Такой механизм получил название денормализации.

***Денормализация** — это процесс достижения компромиссов в нормализованных отношениях посредством намеренного введения избыточности в целях увеличения производительности.*

Фактически необходимость денормализации становится очевидной не в процессе логического проектирования базы данных, а лишь на этапе создания пользовательского приложения, а иногда и в ходе эксплуатации информационной системы. Например, разработчик может заметить, что добавление некоторых дополнительных полей в таблицу БД позволяет значительно сократить время формирования некоторых отчетов.

На рисунке 4.11 представлены два фрагмента схемы БД, содержащей две таблицы — до и после денормализации.

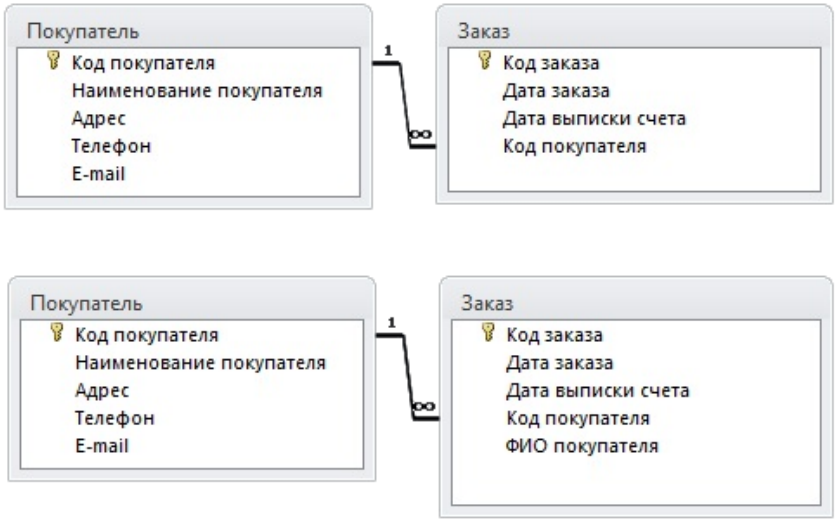


Рис. 4.11 – Пример нисходящей денормализации

Здесь показана так называемая «нисходящая денормализация», когда в таблицу, содержащую внешний ключ, добавляется непосредственно атрибут, который этот внешний ключ определяет (в нашем примере — ФИО покупателя). При такой денормализации в случае выполнения запроса, в котором при формировании данных по счетам необходимо выдать ФИО покупателя, мы избежим выполнения операции соединения таблиц «Заказ» и «Покупатель».

На рисунке 4.12 приведен пример восходящей денормализации.

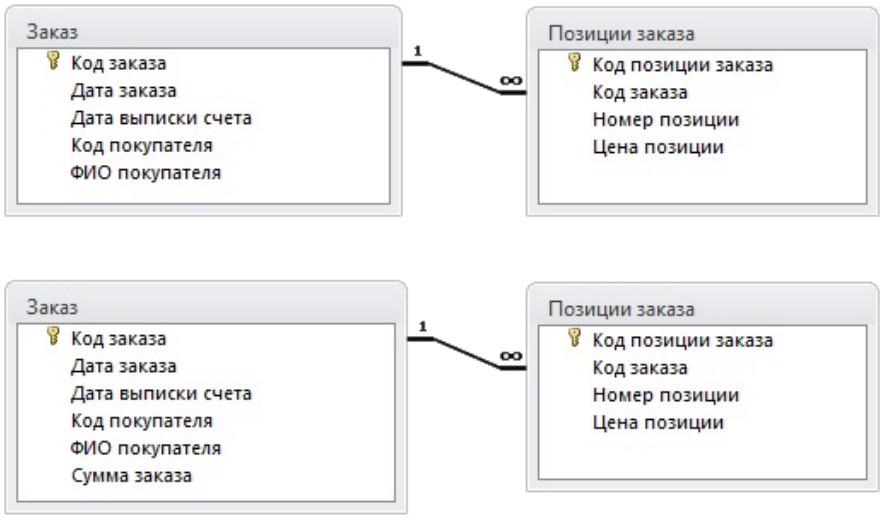


Рис. 4.12 – Пример восходящей денормализации

Здесь в таблицу «Заказ» добавляется поле, содержащее общую стоимость заказа. Тогда, в случае формирования некоторого итогового отчета, мы освобождаемся от выполнения запроса на вычисление суммы всех позиций заказа.

При кажущейся простоте денормализации необходимо учитывать, что ее использование приводит к явным избыточностям в хранении данных, а также может привести к несогласованности информации при ведении базы данных — если содержимое внешних ключей контролируется средствами СУБД, то контроль за содержимым избыточных атрибутов возлагается на разработчика или на непосредственного пользователя информационной системы.

4.2 Моделирование данных с помощью диаграмм «сущность-связь»

4.2.1 Основные понятия модели «сущность-связь»

При проектировании информационных систем иногда трудно моделировать предметную область на основе отношений. Реляционная модель в полной мере не дает представления о семантике 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].

Семантическое моделирование *представляет собой моделирование структуры данных, опираясь на смысл этих данных.*

На начальном этапе анализа предметной области и проектирования БД в идеологии семантической модели проектируется концептуальная модель предметной области, которая затем преобразуется к физической модели и затем к схеме реляционной БД. Этот процесс может выполняться как вручную, так и с помощью различных программных средств. В результате этих преобразований на основе семантической модели производится схема реляционной базы данных в третьей нормальной форме с учетом специфики конкретной СУБД. Чем более разнородна входная информация по структуре и содержанию, чем менее она унифицирована, тем больше внимания необходимо уделять семантическому моделированию.

Мы будем рассматривать одну из разновидностей семантического моделирования, получившую 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 ER-модели нашли широкое применение в CASE-системах, обеспечивающих автоматизированное проектирование БД.

Необходимо отметить, что одна нотация представления ER-диаграмм может несколько отличаться от другой. Однако термины и элементы, которыми опериру-

ют при разработке модели, присутствуют всегда и несут идентичную смысловую нагрузку. В настоящее время наиболее распространенной нотацией, используемой в средствах автоматизированного проектирования, являются нотация Баркера (и ее производные), а также нотация IDEF1X.

Мы будем рассматривать примеры, представленные в нотации, используемой при создании концептуальной модели данных (*Conceptual data model*) в среде автоматизированного моделирования PowerDesigner — она представляет собой расширение нотации Баркера — нотации *Information engineering* и *Merise*.

Заметим, что проектировщики могут самостоятельно настроить в среде PowerDesigner варианты представления ER-диаграмм.

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

Сущность *— это объект предметной области, элемент информационной системы или, другими словами, класс однотипных объектов, информация о которых должна быть учтена в модели. В ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности и наименования ее атрибутов.*

Каждая сущность должна иметь имя, выраженное существительным в единственном числе (Студент, Адрес, Сотрудник) (рис. 4.13).

Рис. 4.13 – Пример графического изображения сущности «Студент»

Экземпляр сущности *— это конкретный представитель данной сущности.*

Например, представителем сущности «Студент» может быть «Студент Иванов».

Атрибут сущности *— это именованная характеристика моделируемого сущностью объекта, являющаяся некоторым свойством сущности. Наименование атрибута должно быть выражено существительным в единственном числе (возможно, с характеризующими прилагательными).*

Атрибут может быть либо обязательным, либо необязательным. Обязательность атрибута означает, что атрибут не может принимать неопределенных значений (null values) (рис. 4.14).

Рис. 4.14 – Сущность «Студент» с набором атрибутов

Атрибут может быть либо описательным (т. е. обычным дескриптором сущности), либо входить в состав уникального идентификатора сущности. При создании ER-модели необходимо учитывать требование уникальности экземпляров сущности, т. е. каждая сущность должна обладать уникальным идентификатором — минимально допустимым набором атрибутов, однозначно характеризующих экземпляр сущности. На рисунке 4.15 представлена сущность «Студент» с набором атрибутов, часть из которых является обязательными (<M> — mandatory), атрибут «№ зачетной книжки» является уникальным идентификатором сущности. Отметим, что в среде PowerDesigner на этапе концептуального моделирования для каждого атрибута задается тип данных.

Рис. 4.15 – Сущность «Студент» с набором атрибутов

В концептуальной модели данных, представленной в виде ER-диаграммы, сущности могут быть связаны между собой.

Связь *— это графическая ассоциация между сущностями.*

Одна сущность может быть связана с другой сущностью или сама с собою. Связи обозначаются линиями с именами, место соединения связи и сущности определяет кардинальность связи (0:1, 1:1, 0:M, 1:M, M:N).

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 (или имеет символ |), а необязательный — прерывистой линией (или имеет символ o). На концах связи указывается имя конца связи, обязательность связи, при необходимости также указывается степень конца связи, то есть количество связываемых экземпляров данной сущности.

На рисунке 4.16 представлен фрагмент ER-диаграммы предметной области ВУЗ, отражающий связь между сущностями «Студент» и «Задолженность за обучение», которая связывает студентов с их задолженностями. 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 обозначается 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.

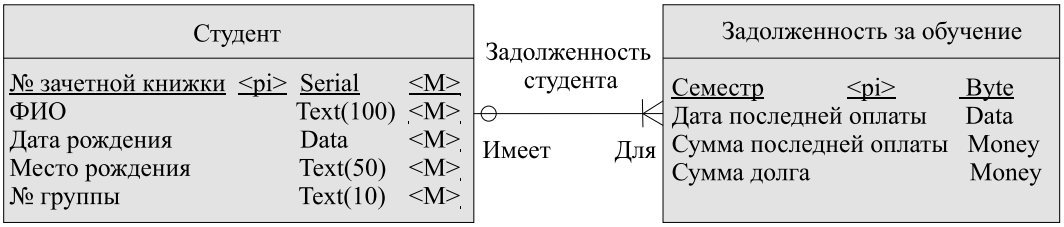


Рис. 4.16 – Пример связи между сущностями ER-модели

4.2.2 Принцип нормализации ER-диаграмм

Нормальные формы ER-диаграмм 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.

4.2.3 Дополнительные элементы ER-модели

В ряде случаев при проектировании модели предметной области основных понятий ER-модели недостаточно, тогда используются 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.

4.2.4 Получение схемы реляционной базы данных из ER-диаграммы

Как мы отмечали выше, большинство средств проектирования ER-диаграмм, среди которых можно выделить такие пакеты, как PowerDesigner, 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. фактически процесс перехода от ER-диаграммы к схеме реляционной базы данных, 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 поле таблицы (атрибут отношения) с тем же именем. Поля таблицы, соответствующие необязательным атрибутам сущности, могут содержать неопределенные значения, тогда как поля, 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 поля являются внешним ключом. Необязательные связи соответствуют полям, допускающим неопределенные значения, обязательные связи — полям таблицы, не допускающим неопределенные значения.

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

На рисунке 4.17 представлен фрагмент физической модели данных, сгенерированной на основе концептуальной модели (рис. 4.16), и схемы БД в СУБД MS Access, сгенерированной на основе этой физической модели. Заметим, что здесь произошла миграция уникального идентификатора «№ зачетной книжки» сущности «Студент» в сущность «Задолженность за обучение», где этот атрибут вошел в состав уникального идентификатора и стал также внешним ключом. Таким образом, очевидно, что при создании концептуальной модели с помощью средств моделирования внешние ключи создавать нет необходимости — они будут сгенерированы автоматически в физической модели, а затем и в схеме базы данных.

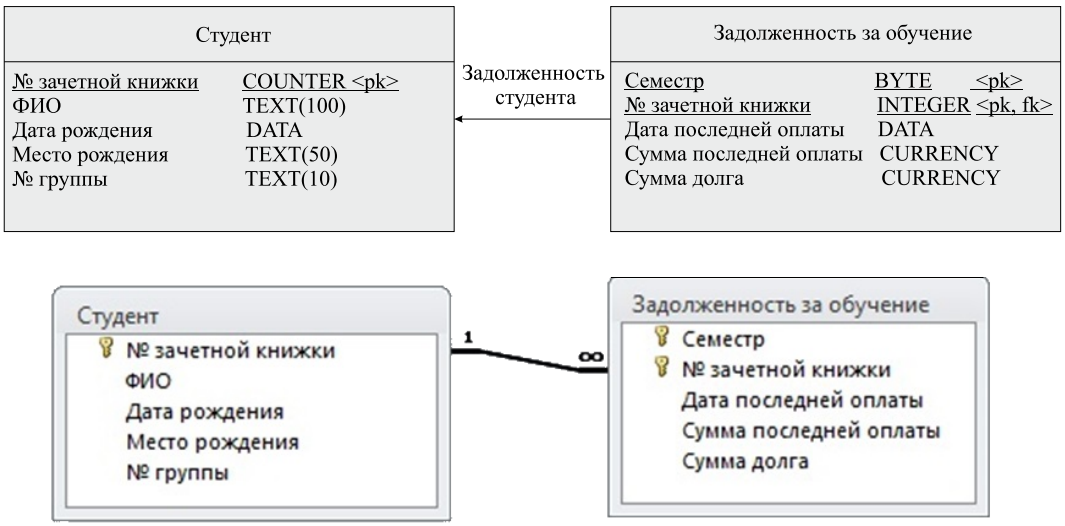


Рис. 4.17 – Пример сгенерированной физической ER-модели и схемы БД

Очевидно, что использование средств автоматизированного моделирования способно значительно облегчить труд разработчиков баз данных. Современные средства моделирования позволяют не только проектировать ER-диаграммы различных уровней сложности, но и позволяют вести процесс эволюции моделей данных с последующей эволюцией схем БД без разрушения созданных ранее объектов БД. Также возможно построение глоссария модели данных с описанием всех объектов, созданных в модели. Кроме этого, существует возможность реинжиниринга модели данных — когда на основе схемы БД строятся физическая и концептуальная ER-диаграммы — это бывает очень полезно в случае необходимости модифицировать созданную ранее БД, для которой отсутствует ее описание.

4.3 CASE-средства

4.3.1 Назначение и классификация CASE-средств

Как отмечалось выше, 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 этом разделе мы остановимся на тех частях CASE-средств, которые отвечают за проектирование структурной части информационных систем, а именно за построение концептуальной и физической моделей данных.

Большинство существующих на рынке CASE-средств 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 информационных систем (*toolkit*), и полностью интегрированные средства, поддерживающие весь жизненный цикл информационных систем и связанные общим репозиторием [12]. По типам CASE-средства можно классифицировать следующим образом [12]:

- CASE-средства, предназначенные для анализа и проектирования. Результатом использования CASE-средств этого типа 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-средства, назначение которых состоит в построении и анализе концептуальной модели предметной области (PowerDesigner (Sybase), 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 (AllFusion ERwin Data Modeler, S-Designor (Powersoft), DataBase Designer (Oracle));
- CASE-средства реинжиниринга — наиболее современные CASE-средства, предоставляющие возможности проведения широкомасштабного анализа программных кодов и схем баз данных и формирования на их основе различных моделей и предварительных проектных спецификаций (Rational Rose (Rational Software), Object Team (Cayenne)).

4.3.2 Обзор CASE-средств

Рассмотрим наиболее часто встречающиеся на российском рынке CASE-средства, в функции которых входит создание концептуальной и физической моделей данных и проектирование на их основе схем 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 информационных систем бизнес-класса. Оно ориентировано в большей степени на спиральную модель ЖЦ, а также 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 проверить 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 БД.

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, SQL-Base, 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* имеются 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:

- проектирования диаграмм потоков данных, ER-диаграмм, структур данных, структурных схем программ и последовательностей экранных форм;

- проектирования диаграмм архитектуры системы (SAD-проектирования состава и связи вычислительных средств, распределения задач системы между вычислительными средствами, моделирования отношений типа «клиент-сервер», анализа использования менеджеров транзакций и особенностей функционирования систем в реальном времени);
- генерации кода программ на языке 4GL целевой СУБД с полным обеспечением программной среды и генерации SQL-кода для создания таблиц БД, индексов, ограничений целостности и хранимых процедур;
- программирования на языке C со встроенным SQL;
- управления версиями и конфигурацией проекта;
- многопользовательского доступа к репозиторию проекта;
- генерации проектной документации по стандартным и индивидуальным шаблонам;
- экспорта и импорта 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.

AllFusion ERwin Data Modeler от разработчика *Computer Associates* предназначено для моделирования и создания баз данных произвольной сложности на основе диаграмм «сущность-связь». В настоящее время *AllFusion ERwin Data Modeler* 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, в пакете 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 вертикального и горизонтального балансирования диаграмм. Так, функциональная и информационная модели сильно коррелированны, что позволяет избавиться от лишних объектов моделей.

PowerDesigner компании *Sybase* — средство моделирования масштаба предприятия, объединяющего технологический и бизнес-уровни моделирования для обеспечения максимально эффективного взаимодействия между бизнес- и IT-пользователями.

В состав *PowerDesigner* 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. Эти элементы передаются на следующий этап проектирования, причем хранилища данных могут быть автоматически преобразованы в 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-ти 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.

Начиная с версии 9.5, *PowerDesigner* позволяет согласовывать объектно-ориентированную и концептуальную модели данных, ориентированную на реляционные СУБД.

PowerDesigner 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.* — продукт, предназначенный для автоматизиции всех этапов проектирования сложных систем различного назначения: формулировки требований и целей проектирования, разработки спецификаций, определения компонентов и взаимодействий между ними, документирования проекта, проверки 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 пять типов отчетов, поддерживающих процесс разработки и анализа моделей.

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

В последнее время CASE-средства становятся все более унифицированными, появляются дополнительные возможности по реализации сложных многоуровневых моделей, механизмы реинжиниринга и анализа моделей. Аналитики и разработчики баз данных имеют возможность выбора — купить мощное средство моделирования (PowerDesigner, ERwin и др.) или воспользоваться бесплатными, свободно распространяемыми продуктами, но с меньшим функционалом (MySQL Workbench, Dia, SQLyog).

Контрольные вопросы по главе 4

1. Дайте определения основных понятий реляционной модели данных.
2. Приведите требования удовлетворения отношений нормальным формам (1NF, 2NF, 3NF).
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.

Глава 5

ЯЗЫКИ УПРАВЛЕНИЯ И МАНИПУЛИРОВАНИЯ ДАННЫМИ

5.1 Язык SQL

5.1.1 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 процессе эволюции СУБД стало очевидно, что любой ***язык, используемый для управления данными в реляционной базе данных****, должен предоставлять разработчику и пользователю следующие основные функциональные возможности:*

- *создавать в базе данных объекты с возможностью полноценного описания их структурной составляющей;*
- *формировать запросы различных уровней сложности на выборку необходимых данных из таблиц БД;*
- *формировать запросы к базе данных, с помощью которых обеспечивается выполнение основных операций по манипулированию данными (вставка, модификация и удаление данных из таблиц).*

Кроме выделенных функциональных возможностей, язык, предназначенный для работы с реляционными базами данных, должен позволять обеспечивать их выполнение с минимальными усилиями со стороны разработчика, при этом син-

таксис команд этого языка и процесс формирования запросов должны быть достаточно просты и доступны для изучения разработчикам разных уровней подготовки.

Также, беря во внимание большое количество СУБД, существующих на рынке, такой язык должен быть в меру универсальным, соответствовать некоторому стандарту, что позволит использовать один и тот же синтаксис и структуру команд (с минимальными изменениями) при написании запросов в разных СУБД.

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

- язык структурированных запросов ***SQL*** (*Structured Query Language*), созданный фирмой IBM в начале 70-х годов XX века. В первоначальном варианте язык получил название ***SEQUEL*** (*Structured English Query Language*) и был разработан для созданной фирмой IBM СУБД System/R. Позднее язык получил свое современное название — SQL;
- 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.

Первыми СУБД, поддерживающими SQL, стали СУБД Oracle V2 в 1979 году от компании *Relational Software Inc*. (впоследствии ставшей компанией *Oracle*) и System/38 от IBM, основанная на базе СУБД System/R.

Помимо операторов формулирования запросов к базе данных и возможностей манипулирования данными, язык SQL содержит большое количество функциональных средств, например:

- 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.

5.1.2 Стандарты языка SQL

С течением времени выявилась основная проблема использования языка SQL — у разных производителей СУБД используются разные диалекты SQL, часто между собой несовместимые. Так, запрос на выборку, созданный и прекрасно выполняющийся в СУБД MS Access, может выдать ошибки и не запускаться в СУБД Oracle, и наоборот.

Фактически, различные СУБД предоставляют возможность выполнения программных конструкций, записанных в разных диалектах и процедурных расширениях (дополнительных возможностях, позволяющих, например, реализовать процедуры и функции) языка SQL. Существует достаточное количество публикаций по различным диалектам и процедурным расширениям языка SQL. Отметим, что

первый официальный стандарт языка SQL был принят ANSI (*American National Standards Institute*) в 1986 году и ISO (Международной организацией по стандартизации) в 1987 году (названный SQL-86).

До 1996 года вопросами соответствия диалектов SQL в конкретных СУБД официальному стандарту занимался в основном Национальный институт стандартов и технологий (NIST). Начиная со стандарта SQL-92, ANSI и NIST определили четыре уровня соответствия конкретной реализации SQL этому стандарту:

- *Entry* (базовый);
- *Transitional* (переходный) — проверку на соответствие этому уровню проводил только институт NIST;
- *Intermediate* (промежуточный);
- *Full* (полный).

Любая компания-разработчик СУБД, в случае, если в этой СУБД поддерживался синтаксис команд уровня Entry, могла заявлять себя как «SQL-92 compliant» — соответствующей стандарту SQL-92, хотя на самом деле переносимость и соответствие стандарту ограничивалось набором возможностей выполнения запросов, входящих в этот базовый уровень.

С появлением и утверждением стандарта SQL:1999 стандарт языка SQL (ANSI SQL) приобрел модульную структуру, в которой основная часть стандарта была вынесена в раздел «SQL/Foundation», а все остальные были выведены в отдельные модули. В таблице 5.1 представлены основные версии стандартов языка SQL [16].

Таблица 5.1 – Основные версии стандартов SQL

Год утверждения	**Название**	**Альтернативное название**	**Описание изменений стандарта**
1986	SQL-86	SQL-87	Первый вариант стандарта, принятый институтом ANSI и одобренный ISO в 1987 году
1989	SQL-89	FIPS 127-1	Небольшое изменение, в котором важным дополнением были ограничения целостности
1992	SQL-92	SQL2, FIPS 127-2	Значительные изменения (ISO 9075); уровень Entry Level стандарта SQL-92 принят как стандарт FIPS 127-2
1999	SQL:1999	SQL3	Добавлена поддержка регулярных выражений, рекурсивных запросов (например, транзитивное замыкание), поддержка триггеров, базовые процедурные расширения, нескалярные типы данных и некоторые объектно-ори-

продолжение на следующей странице

Таблица 5.1 — Продолжение

Год утверждения	Название	Альтернативное название	Описание изменений стандарта
			ентированные возможности (например, структурные типы). Поддержка для встраивания SQL в Java, и наоборот
2003	SQL:2003	SQL2003	Введены расширения для работы с XML-данными, оконные функции (применяемые для работы с OLAP-базами данных), генераторы последовательностей и основанные на них типы данных атрибутов таблиц
2006	SQL:2006	SQL2006	Функциональность работы с XML-данными значительно расширена. Определены способы импорта и хранения XML-данных в базе данных SQL. Появилась возможность совместно использовать в запросах SQL и XQuery (язык запросов XML)
2008	SQL:2008	SQL2008	Улучшены возможности оконных функций, устранены некоторые неоднозначности стандарта SQL:2003
2011	SQL:2011		Улучшены функции по работе с временными базами данных

Разнообразие использования различных диалектов процедурного расширения стандарта языка ANSI SQL (SQL/PSM — SQL/Persistent Stored Modules) в наиболее известных СУБД представлено в таблице 5.2.

Таблица 5.2 – Основные версии стандартов SQL

СУБД	Сокращенное наименование	Полное наименование и описание
Interbase/Firebird	PSQL	Procedural SQL
IBM DB2	SQL PL	SQL Procedural Language (реализованы базовые возможности SQL/PSM)
IBM Informix	SPL	Stored Procedural Language
IBM Netezza	NZPLSQL	Основан на Postgres PL/pgSQL
Microsoft/Sybase	T-SQL	Transact-SQL
Mimer SQL	SQL/PSM	SQL/Persistent Stored Module (реализованы базовые возможности SQL/PSM)

продолжение на следующей странице

Таблица 5.2 — Продолжение

СУБД	Сокращенное наименование	Полное наименование и описание
MySQL	SQL/PSM	SQL/Persistent Stored Module (реализованы базовые возможности SQL/PSM)
NuoDB	SSP	Starkey Stored Procedures
Oracle	PL/SQL	Procedural Language/SQL (основан на синтаксисе языка Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL (основан на языке Oracle PL/SQL)
PostgreSQL	PL/PSM	Procedural Language/Persistent Stored Modules (реализованы базовые возможности SQL/PSM)
Sybase	Watcom-SQL	SQL Anywhere Watcom-SQL Dialect
Teradata	SPL	Stored Procedural Language
SAP	SAP HANA	SQL Script

5.1.3 Описание основных команд SQL

Что касается синтаксиса запросов и предложений, написанных на языке SQL, то в большинстве работ по описанию команд SQL авторы придерживаются общего синтаксиса написания SQL-предложений:

1) в описании команд слова, написанные прописными латинскими буквами, являются зарезервированными словами SQL;
2) фрагменты SQL-предложений, заключенные в фигурные скобки и разделенные символом «|», являются альтернативными. При формировании соответствующей команды для конкретного случая необходимо выбрать одну из них;
3) фрагмент описываемого SQL-предложения, заключенный в квадратные скобки **[ ]**, имеет необязательный характер и может не использоваться;
4) многоточие перед закрывающейся скобкой говорит о том, что фрагмент, указанный в этих скобках, может быть повторен.

На рисунке 5.1 схематично представлен SQL-запрос на выборку данных из таблицы «Студент».

Мы остановимся на описании основных команд языка SQL в диалекте языка, используемого в СУБД MS Access (Microsoft Jet SQL), который имеет некоторые отличия от стандарта ANSI SQL, о которых будет сказано ниже. Однако освоив создание SQL запросов в одной СУБД, вы без особых усилий сможете применить свои знания при реализации запросов в других СУБД.

Создание новой таблицы

*Инструкция **CREATE TABLE** создает новую таблицу и используется для описания ее полей и индексов.*

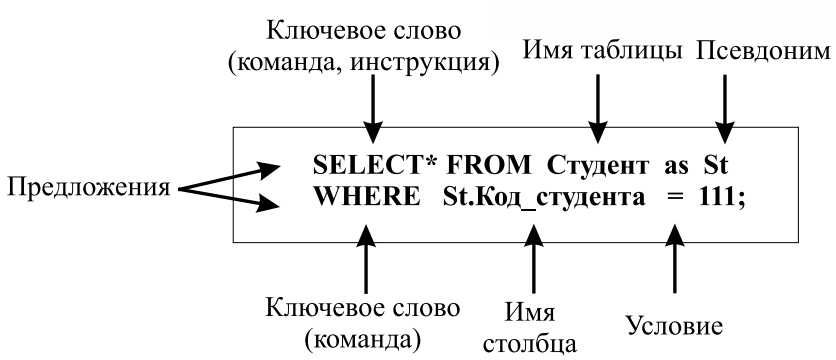


Рис. 5.1 – Схематичное представление SQL-запроса

Если для поля добавлено ограничение NOT NULL, то при добавлении новых записей это поле не должно содержать несуществующих (NULL) данных.

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 представлено создание новой таблицы «Студент».

Пример 5.1

CREATE TABLE Студент (Код_студента AUTOINCREMENT PRIMARY KEY, Номер_зачетной_книжки INTEGER, ФИО_студента TEXT (50), Место_рождения TEXT (50));

В результате выполнения этого запроса в БД СУБД MS Access будет создана таблица «Студент», представленная в схеме БД следующим образом (рис. 5.2).

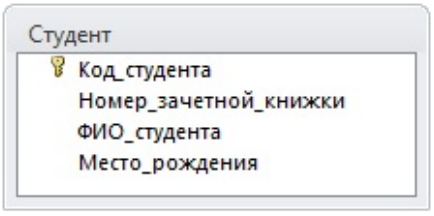


Рис. 5.2 – Таблица «Студент» в схеме данных

На рисунке 5.3 таблица «Студент» представлена в режиме конструктора таблиц. Отметим, что в таблицу добавлен суррогатный первичный ключ «Код_студента», имеющий тип данных «Счетчик» или AUTOINCREMENT, — этот тип данных специально используется для автоматического формирования значений суррогатных первичных ключей — атрибут, определенный на таком типе данных, будет принимать уникальные числовые значения при создании новой записи в таблице.


Студент

Имя поля	Тип данных
Код_студента	Счетчик
Номер_зачетной_книжки	Числовой
ФИО_студента	Текстовый
Место_рождения	Текстовый

Рис. 5.3 – Таблица «Студент» в режиме конструктора таблиц

*Предложение **CONSTRAINT** используется в инструкциях ALTER TABLE и CREATE TABLE для создания или удаления индексов.*

Индексы — это специальные объекты в БД, создаваемые с целью повышения быстродействия выполнения запросов, оптимизации хранения и доступа к данным. Об индексах будет рассказано в следующем разделе учебного пособия. 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]
[ON UPDATE {CASCADE | SET NULL}]
[ON DELETE {CASCADE | SET NULL}]}

- составной индекс:

CONSTRAIN T 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 [,. . .]])]
[ON UPDATE {CASCADE | SET NULL}]
[ON DELETE {CASCADE | SET NULL}]},

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 внешнее Поле; 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.

ON UPDATE, ON DELETE обеспечивают автоматическое указание каскадного обновления связанных полей и каскадного удаления связанных записей в схеме БД (CASCADE) или обнуление соответствующих значений полей, являющихся внешними ключами (SET NULL). Данные ключевые слова актуальны, если в СУБД MS Access включить поддержку ANSI SQL, в противном случае в результате выполнения запроса будет выдана ошибка синтаксиса.

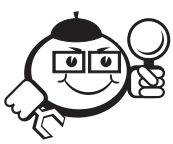
Предложение CONSTRAINT позволяет создать для поля индекс одного из двух описанных ниже типов:

1) уникальный индекс, создаваемый с помощью зарезервированного слова 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) ключ таблицы, состоящий из одного или нескольких полей, использующий 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 внешним ключом «Код_студента», связанным с полем «Код_студента», в таблице «Студент».



Пример 5.2

```
CREATE TABLE Задолженность_за_обучение
(Код_задолженности AUTOINCREMENT PRIMARY KEY,
```

Код_студента INTEGER, Сумма_задолженности MONEY,
CONSTRAINT f1_i FOREIGN KEY (Код_студента)
REFERENCES Студент (Код_студента)
ON UPDATE CASCADE ON DELETE CASCADE);

Внешний вид схемы БД, состоящей из таблиц «Студент» и «Задолженность_за_обучение», представлен на рисунке 5.4. Здесь необходимо отметить, что дополнительные ключевые слова ON UPDATE CASCADE ON DELETE CASCADE позволили обеспечить автоматическое указание каскадного обновления связанных полей и каскадного удаления связанных записей в схеме БД.

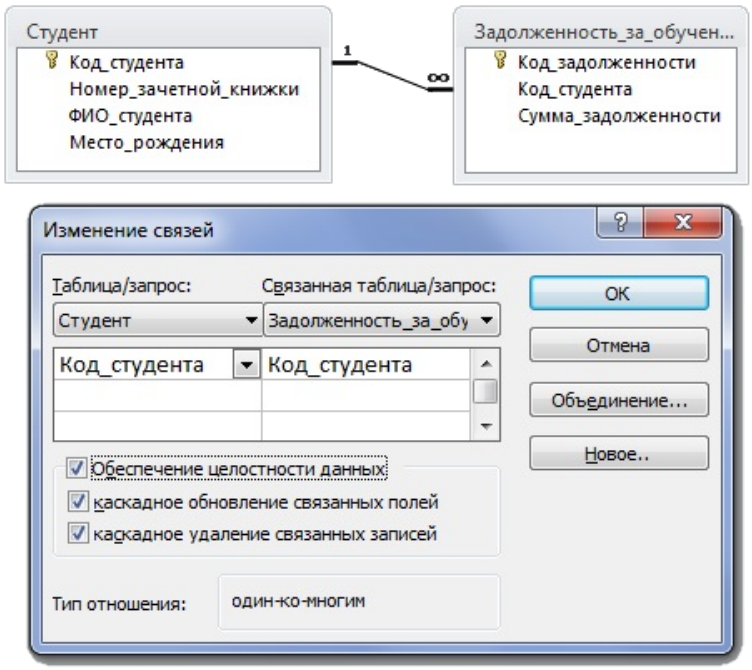


Рис. 5.4 – Схема данных после создания таблицы «Задолженность_за_обучение»

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. 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.

В следующем примере в таблицу «Студент» будет добавлено поле Дата_рождения (рис. 5.5).

Пример 5.3

ALTER TABLE Студент ADD COLUMN Дата_рождения date;

Студент

Имя поля	Тип данных
Код_студента	Счетчик
Номер_зачетной_книжки	Числовой
ФИО_студента	Текстовый
Место_рождения	Текстовый
Дата_рождения	Дата/время

Рис. 5.5 – Результат добавления поля «Дата_рождения» в таблицу «Студент»

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 назначения. Например:

1) с помощью параметра DISALLOW NULL можно запретить значения NULL в индексированных полях новых записей;
2) параметр IGNORE NULL позволяет запретить включение в индекс записей, имеющих значения NULL в индексированных полях;
3) зарезервированное слово PRIMARY позволяет назначить индексированные поля ключом. Такой индекс по умолчанию является уникальным, следовательно, зарезервированное слово UNIQUE можно опустить.

В следующем примере создается уникальный индекс, не допускающий ввод повторяющихся значений в поле «Номер зачетной книжки» в таблице «Студент».

Пример 5.4

CREATE UNIQUE INDEX New_index ON
Студент (Номер_зачетной_книжки) WITH IGNORE NULL;

Ключевые слова IGNORE NULL позволяют добавлять в таблицу записи по студентам без обязательного ввода номера зачетной книжки (рис. 5.6). Нажатие пиктограммы «Индексы» в конструкторе таблиц открывает одноименное окно с перечнем индексов для данной таблицы, где отображается созданный в результате выполнения запроса уникальный индекс New_index.

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.

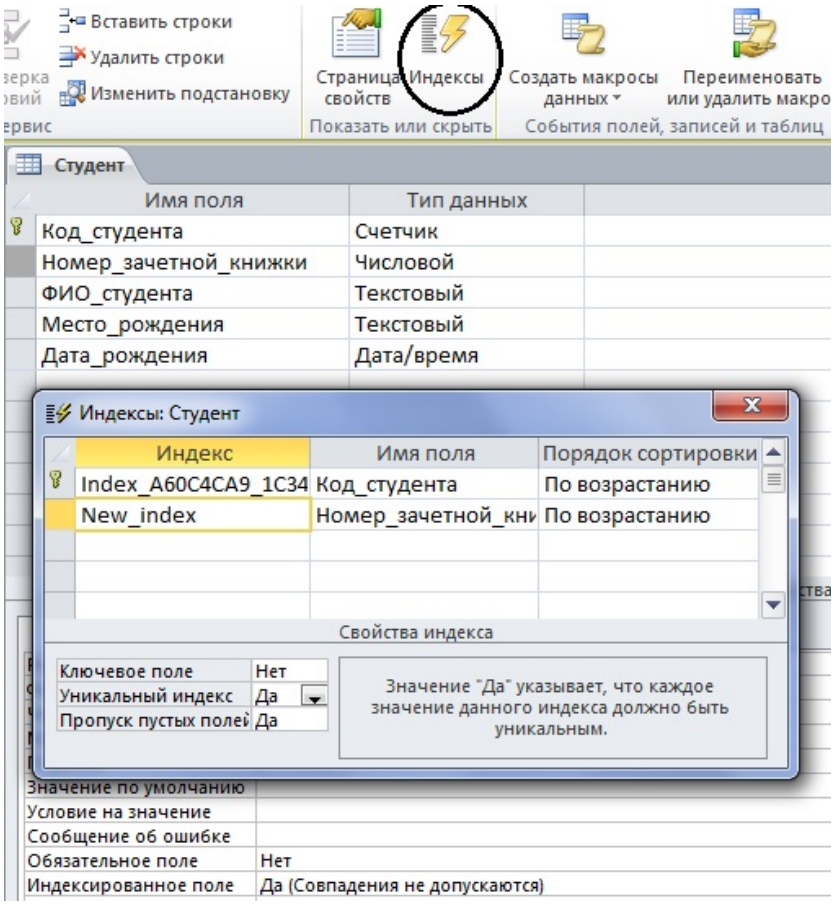


Рис. 5.6 – Результат создания уникального индекса

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

В следующем примере удалим созданный в предыдущем примере индекс «New_index».

Пример 5.5

```
DROP INDEX New_index ON Студент;
```

В результате выполнения этого запроса в окне Индексы (рис. 5.7) остался только уникальный индекс, созданный автоматически при определении первичного ключа.

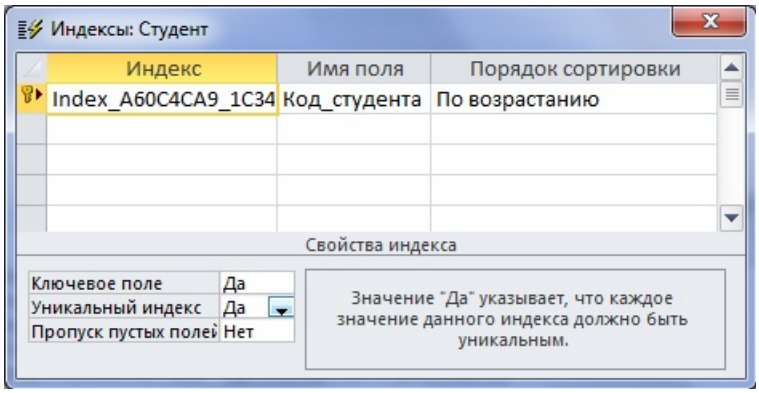


Рис. 5.7 – Окно «Индексы» после удаления индекса «New_index»

Добавление записей

*Инструкция **INSERT INTO** добавляет запись или несколько записей в заданную 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 операций 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.

В следующем примере добавим новую запись в таблицу «Студент».

Пример 5.6

```
INSERT INTO Студент (Номер_зачетной_книжки, ФИО_студента, Место_рождения, Дата_рождения)
VALUES (201454321, 'Иванов Иван Петрович', 'г. Томск', '12.02.1996');
```

Заметим, что поскольку поле «Код_студента» имеет тип данных «Счетчик» и формируется автоматически, то в перечне новых значений для полей мы его опускаем. На рисунке 5.8 представлена таблица «Студент» с новой записью.

Студент

Код_студента	Номер_зачетной_книжки	ФИО_студента	Место_рождения	Дата_рождения
1	201454321	Иванов Иван Петро	г. Томск	12.02.1996

Рис. 5.8 – Результат добавления записи в таблицу «Студент»

Аналогично можно добавить данные в таблицу «Задолженность_за обучение».

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.

Пример 5.7

Следующая инструкция SQL увеличивает сумму задолженности всех студентов, по которым есть сведения в таблице «Задолженность_за_обучение», на 10%:

```
UPDATE Задолженность_за_обучение
SET Сумма_задолженности = Сумма_задолженности * 1.1;
```

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 выбираются 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.

Для более наглядного представления результатов выполнения запросов на выборку добавим в нашу базу данных таблицы «Дисциплина» и «Успеваемость» и добавим поле «Номер_группы» в таблицу «Студенты» (рис. 5.9), а также заполним эти таблицы.

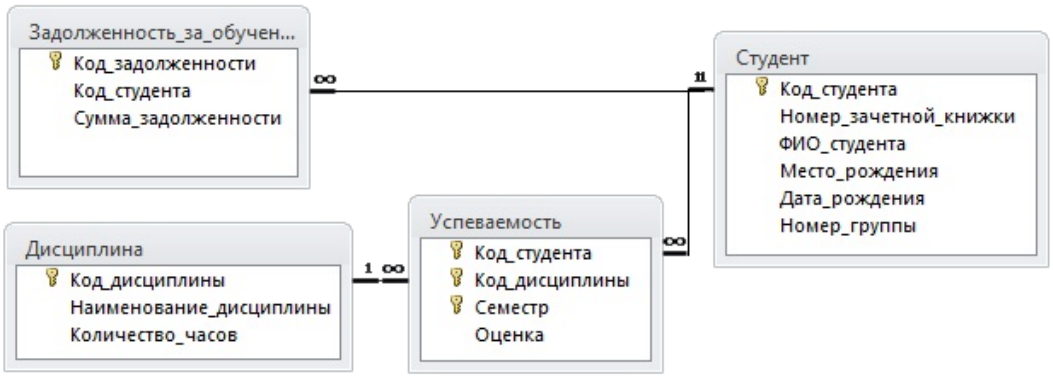


Рис. 5.9 – Обновленная схема базы данных

На рисунке 5.10 приведено содержимое таблиц «Студент», «Успеваемость» и «Дисциплина».

В следующем примере представлен простейший запрос на выборку записей из таблицы «Студенты», в котором отбираются студенты, обучающиеся в группе 432-1.

Пример 5.8

```
SELECT Студент.Номер_зачетной_книжки,
       Студент.ФИО_студента, Студент.Номер_группы
FROM Студент
WHERE Студент.Номер_группы='432-1';
```

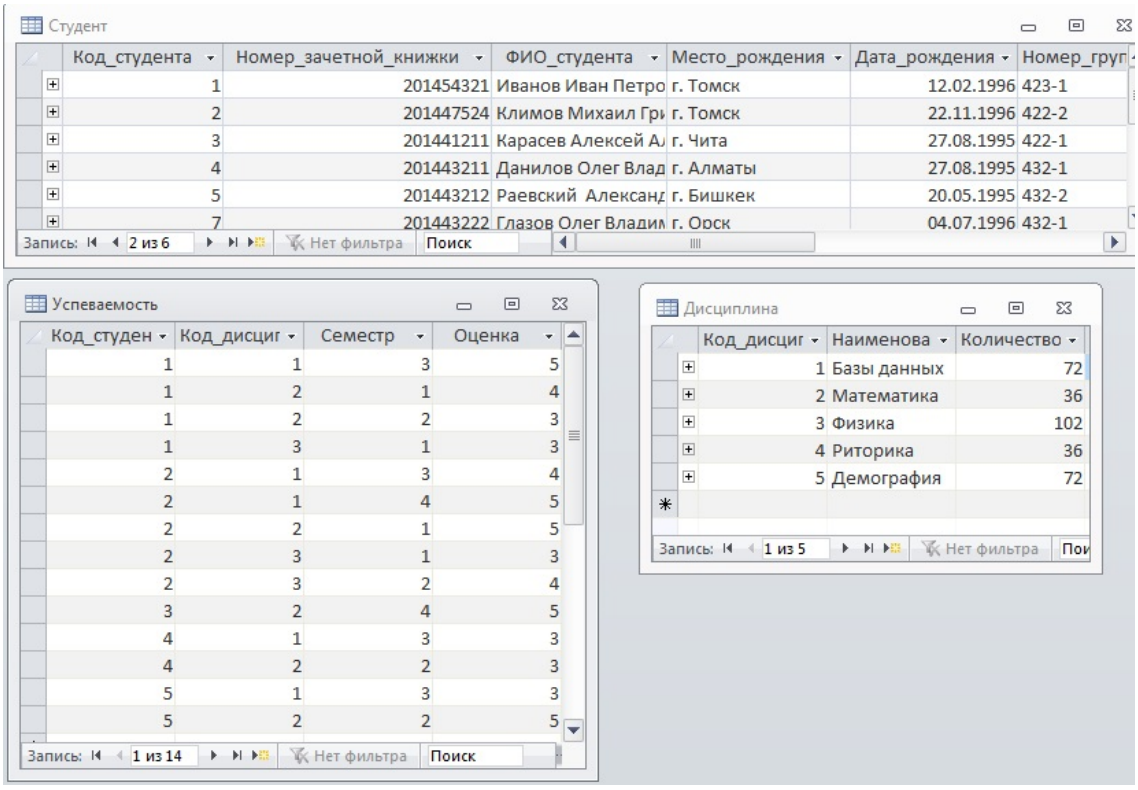


Рис. 5.10 – Содержимое таблиц базы данных

На рисунке 5.11 представлен результат выполнения этого запроса, были найдены две записи, удовлетворяющие критерию запроса.

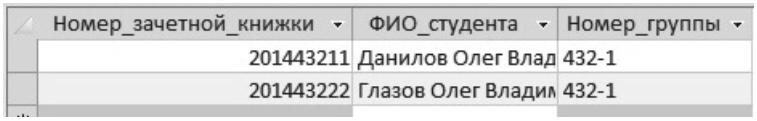


Рис. 5.11 – Результат выполнения запроса

Помимо обычных знаков сравнения (=, <, >, <=, >=, <>) в языке SQL в условии отбора используется 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), то 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.

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 5.3 приведен перечень подстановочных символов и пример их использования в диалекте языка SQL, используемого в MS Access (Microsoft Jet SQL).

Таблица 5.3 – 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.

В следующем примере составим запрос, выдающий сведения об успеваемости студентов по всем изученным ими дисциплинам.

Пример 5.9

```
SELECT Студент.ФИО_студента as ФИО, Дисциплина.Наименование_дисциплины as Предмет, Успеваемость.Семестр, Успеваемость.Оценка
FROM Студент INNER JOIN (Дисциплина INNER JOIN Успеваемость ON Дисциплина.Код_дисциплины = Успеваемость.Код_дисциплины)
ON Студент.Код_студента = Успеваемость.Код_студента;
```

На рисунке 5.12 представлен результат выполнения этого запроса.

ФИО_студента	Наименова	Семестр	Оценка
Иванов Иван Петрович	Базы данных	3	5
Иванов Иван Петрович	Математика	1	4
Иванов Иван Петрович	Математика	2	3
Иванов Иван Петрович	Физика	1	3
Климов Михаил Григорьевич	Базы данных	3	4
Климов Михаил Григорьевич	Базы данных	4	5
Климов Михаил Григорьевич	Математика	1	5
Климов Михаил Григорьевич	Физика	1	3
Климов Михаил Григорьевич	Физика	2	4
Карасев Алексей Александрови	Математика	4	5
Данилов Олег Владимирович	Базы данных	3	3
Данилов Олег Владимирович	Математика	2	3
Раевский Александр Иванович	Базы данных	3	3
Раевский Александр Иванович	Математика	2	5

Запись: 2 из 14 | Нет фильтра | Поиск

Рис. 5.12 – Результат выполнения запроса с внутренним соединением

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.

В следующем примере в результате выполнения запроса будут найдены все студенты и их успеваемость, при этом в результирующий набор данных будут также включены студенты, чьи сведения об успеваемости отсутствуют.

Пример 5.10

SELECT Студент.ФИО_студента, Дисциплина.Наименование_дисциплины, Успеваемость.Семестр, Успеваемость.Оценка
FROM Студент LEFT JOIN (Дисциплина RIGHT JOIN Успеваемость ON Дисциплина.Код_дисциплины = Успеваемость.Код_ дисциплины)
ON Студент.Код_студента = Успеваемость.Код_студента;

На рисунке 5.13 представлен результат выполнения этого запроса, где видно, что по сравнению с предыдущим примером добавлена еще одна запись о студенте Глазове Олеге Владимировиче.

ФИО_студента	Наименова	Семестр	Оценка
Иванов Иван Петрович	Базы данных	3	5
Иванов Иван Петрович	Математика	1	4
Иванов Иван Петрович	Математика	2	3
Иванов Иван Петрович	Физика	1	3
Климов Михаил Григорьевич	Базы данных	3	4
Климов Михаил Григорьевич	Базы данных	4	5
Климов Михаил Григорьевич	Математика	1	5
Климов Михаил Григорьевич	Физика	1	3
Климов Михаил Григорьевич	Физика	2	4
Карасев Алексей Александрови	Математика	4	5
Данилов Олег Владимирович	Базы данных	3	3
Данилов Олег Владимирович	Математика	2	3
Раевский Александр Иванович	Базы данных	3	3
Раевский Александр Иванович	Математика	2	5
Глазов Олег Владимирович			
*			

Запись: 15 из 15 Нет фильтра Поиск

Рис. 5.13 – Результат выполнения запроса с внешним соединением

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 операции LEFT JOIN или RIGHT JOIN.

Перекрестные запросы

В Jet SQL существует такой вид запросов, как перекрестный. В таком запросе отображаются результаты статистических функций — суммы, средние значения и др., а также количество записей. При этом подсчет выполняется по данным одного из 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 балл за семестр студентов, обучающихся в разных группах, необходимо реализовать перекрестный запрос, результат выполнения которого будет представлен в виде таблицы, где заголовками строк будут служить названия предмета, заголовками столбцов — номера групп, а в полях таблицы будет рассчитан средний балл студентов группы по каждому предмету.

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.

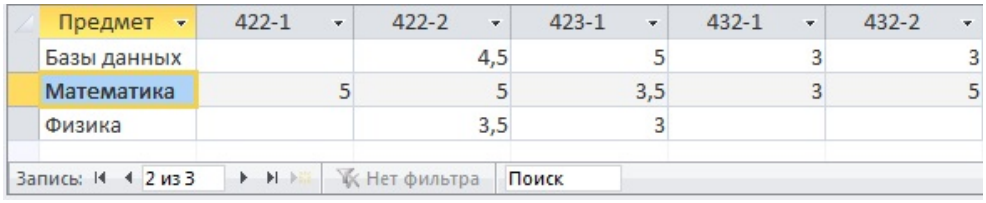
Пример 5.11

Составим SQL-запрос, реализующий описанный выше пример.

```
TRANSFORM Avg(Успеваемость.Оценка)
SELECT Дисциплина.Наименование_дисциплины as [Предмет]
FROM Студент INNER JOIN
(Дисциплина INNER JOIN Успеваемость ON Дисциплина.Код_дисциплины =
Успеваемость.Код_дисциплины) ON Студент.Код_студента = Успеваемость.Код_
студента
GROUP BY Дисциплина.Наименование_дисциплины, Успеваемость.Код_дисциплины
PIVOT Студент.Номер_группы;
```

В результате выполнения такого перекрестного SQL-запроса формируется набор данных, представленный на рисунке 5.14.

Таким образом, когда данные сгруппированы с помощью перекрестного запроса, можно выбирать значения из заданных столбцов или выражений и определять как заголовки столбцов. Это позволяет просматривать данные в более компактной форме, чем при работе с обычным запросом на выборку. Отметим, что перекрестные запросы удобно использовать для формирования статистических отчетов.



Предмет	422-1	422-2	423-1	432-1	432-2
Базы данных		4,5	5	3	3
Математика	5	5	3,5	3	5
Физика		3,5	3		

Запись: 2 из 3 Нет фильтра Поиск

Рис. 5.14 – Результат выполнения перекрестного запроса

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 которых представлен ниже:

1) сравнение [ANY | ALL | SOME] (инструкция SQL);
2) выражение [NOT] IN (инструкция SQL);
3) [NOT] EXISTS (инструкция SQL),

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; инструкция SQL — инструкция SELECT, заключенная в 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.

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, у которых средний балл 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.

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 используем подчиненный запрос на выборку в инструкции UPDATE — увеличим сумму задолженности за обучение не для всех студентов, а только для студента группы 423-1 Иванова Ивана Петровича.

Пример 5.12

```
UPDATE Задолженность_за_обучение
SET Сумма_задолженности = Сумма_задолженности * 1.1
WHERE Код_студента =
(Select Код_студента FROM Студент WHERE ФИО_Студента = 'Иванов Иван Петрович'AND Номер_группы = '423-1');
```

На рисунке 5.15 представлена таблица «Задолженность_ за_обучение» до и после выполнения запроса. (Значение поля «Код_студента» для Иванова Ивана Петровича равно 1.)

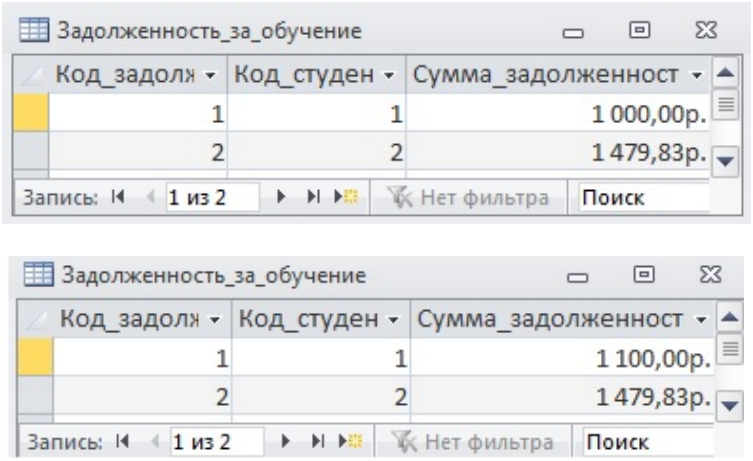
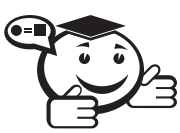


Рис. 5.15 – Результат выполнения запроса на обновление с подчиненным запросом

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.

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.

Пример 5.13

Для выполнения следующего запроса создадим в БД и заполним таблицу «Абитуриент» (рис. 5.16).

Абитуриент

Код_абитуриен	Номер_зачет	ФИО_абитуриент	Место_рождения	Дата_рождения	Номер_групп
8		Иванкова И.С.	г. Томск	11.10.1995	421-1
9		Авдеев Н.В.	г. Омск	01.04.1994	421-1
10		Красников И.И.	г. Бийск	12.02.1997	412-2
*	(№)				

Запись: 1 из 3 Нет фильтра Поиск

Рис. 5.16 – Таблица «Абитуриент»

Здесь объединяются существующая таблица «Студент» и запрос на выборку всех записей из таблицы «Абитуриент».

```
TABLE Студент
UNION ALL SELECT *
FROM Абитуриент;
```

Результат выполнения запроса представлен на рисунке 5.17.

Union

Код_студента	Номер_зачетной_	ФИО_студе	Место_рож	Дата_рожд	Номер_групп
1	201454321	Иванов Иван І	г. Томск	12.02.1996	423-1
2	201447524	Климов Михаі	г. Томск	22.11.1996	422-2
3	201441211	Карасев Алекс	г. Чита	27.08.1995	422-1
4	201443211	Данилов Олег	г. Алматы	27.08.1995	432-1
5	201443212	Раевский Але	г. Бишкек	20.05.1995	432-2
7	201443222	Глазов Олег В.	г. Орск	04.07.1996	432-1
9		Авдеев Н.В.	г. Омск	01.04.1994	421-1
10		Красников И.І	г. Бийск	12.02.1997	412-2
8		Иванкова И.С.	г. Томск	11.10.1995	421-1

Запись: 2 из 9 Нет фильтра Поиск

Рис. 5.17 – Результат выполнения запроса на объединение

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

ровать возврат всех записей. Кроме того, такие запросы выполняются несколько быстрее.

Таблица и все запросы, включенные в операцию UNION, должны отбирать одинаковое число полей, при этом имена полей, типы данных и размеры полей могут не совпадать.

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.

Удаление записей

*Инструкция **DELETE** создает запрос на удаление записей из одной или нескольких таблиц, перечисленных в предложении FROM и удовлетворяющих 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 инструкции 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.

В следующем примере из таблицы «Абитуриент» будет удален абитуриент Авдеев Н. В.

Пример 5.14

```
DELETE
FROM Абитуриент
WHERE ФИО_Абитуриента = 'Авдеев Н. В.'
```

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 условия отбора в предложении Where, а затем выполнить запрос на удаление.

5.1.4 Особые возможности и основные различия языка Microsoft Jet и ANSI SQL

В Microsoft Jet SQL предоставлен ряд функций, обеспечивающих выполнение групповых операций, а также функции работы с датой и временем. Отметим, что некоторые из этих функций также могут быть использованы при написании запросов в других СУБД, а не только в MS Access.

Основные групповые функции, которые можно использовать в запросах на выборку:

- 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;
- Day(дата) возвращает значение дня месяца в диапазоне 1–31;
- Month(дата) возвращает значение месяца года в диапазоне от 1 до 12;
- Year(дата) возвращает значение года в диапазоне 100–9999.

В языке Microsoft Jet SQL поддерживаются следующие дополнительные средства выполнения запросов:

- инструкция TRANSFORM, предназначенная для создания перекрестных запросов;
- дополнительные статистические функции, такие как StDev и VarP;
- описание PARAMETERS, предназначенное для создания запросов с параметрами.

Ниже приведены основные различия двух диалектов языка SQL — Microsoft Jet SQL и ANSI SQL:

- языки SQL-ядра базы данных Microsoft Jet SQL 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 *значению_2*;
- разные подстановочные знаки используются с оператором Like. Так, в языке Microsoft Jet SQL 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 «%».

В языке Microsoft Jet SQL 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, 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.

5.2 Язык Query-by-Example

5.2.1 Основы языка QBE

Говоря о языке SQL, нельзя не упомянуть о языке построения запросов QBE. Основной принцип формирования запросов на языке QBE заключается в том, что запрос на обработку формулируется путем заполнения некоторой пустой таблицы, в основном соответствующей исходной таблице, являющейся источником записей для результирующего набора данных. В таблицу добавляются дополнительные столбцы, если в результирующем наборе данных необходимо наличие столбца, значение которого является константой либо 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 представлены 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: добавление данных I. — Insert.; обновление данных U. — Update.; удаление D. — Delete.

На рисунке 5.18 представлены примеры запросов, созданных 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.	2014432-13	Сидоров А.Н.	05.09.1995	Омск	422-1
U.	2014432-02	Казаков А.Н			
D.					422-3

Рис. 5.18 – Примеры записи запросов средствами QBE

5.2.2 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 (область построения запросов) MS Access или, как его называют, областью QBE.

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.

Пример 5.15

Так, следующий SQL-запрос на выборку может быть представлен в виде бланка построителя запросов в СУБД MS Access (рис. 5.19).

```
SELECT Студент.Номер_зачетной_книжки, Студент.ФИО_студента,
        Студент.Номер_группы
FROM Студент
WHERE (Студент.Номер_группы Like '422*')
AND (Студент.Место_рождения= 'г. Чита');
```

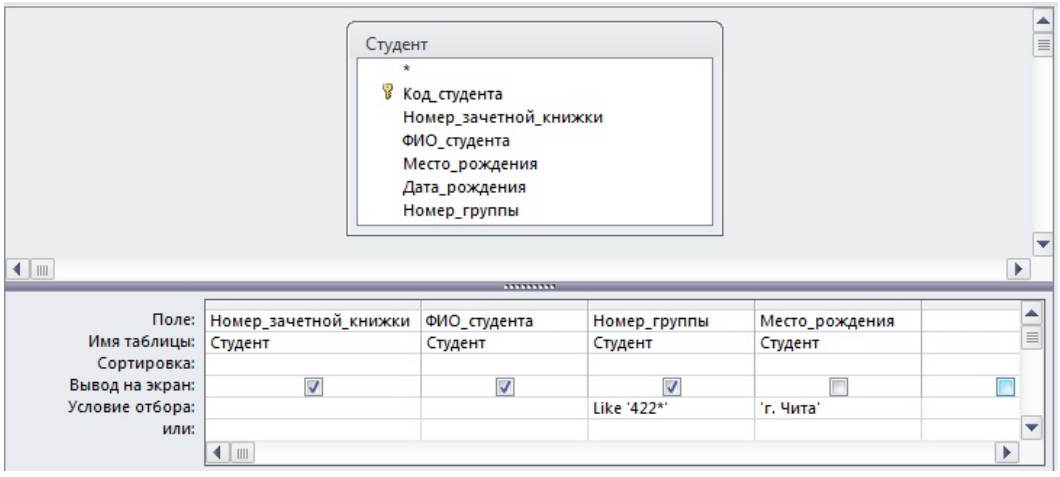


Рис. 5.19 – Пример записи запроса в бланке построения запросов MS Access

В общем случае поля, вводимые в наборе записей запроса, наследуют свойства, заданные для соответствующих полей таблицы. При создании запроса можно задать критерии отбора, 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. Например, в бланке построения запросов можно задать несколько условий отбора, соединенных логическими операторами «и» (AND) или «или» (OR), для одного или нескольких полей.

Также в бланке построения запросов СУБД MS Access можно использовать групповые, математические и другие функции, которые можно задать при группировке или в условиях отбора (например, для даты и времени). Можно задать вычисления над любыми полями таблицы и сделать вычисляемое значение новым полем в запросе.

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 был представлен выше на рисунке 5.14, также может быть реализован с помощью бланка построения запросов (рис. 5.20).

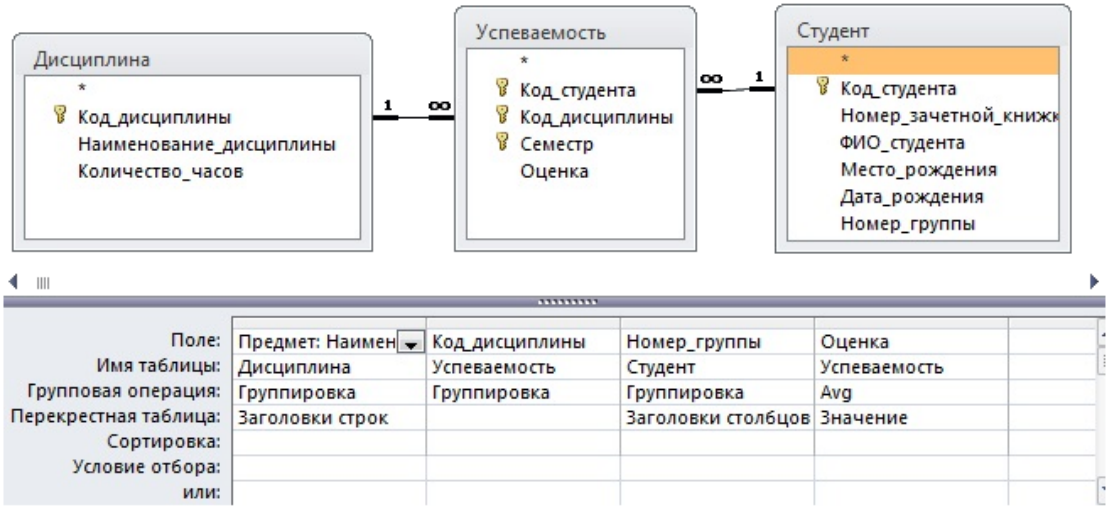


Рис. 5.20 – Перекрестный запрос, созданный с помощью бланка построителей запросов

Следует отметить, что любой запрос, созданный с помощью построителя запросов в среде MS Access, преобразуется в SQL-запрос. Однако не каждый SQL-запрос может быть представлен с помощью бланка построения запросов.

Контрольные вопросы по главе 5

1. Поясните необходимость наличия средств манипулирования данными в СУБД.
2. Приведите примеры SQL-запросов на создание таблиц.
3. Создайте несколько SQL-запросов, используя операции выборки, обновления и удаления данных.
4. Поясните принцип построения запросов в бланке построения запросов MS Access.

Глава 6

ФИЗИЧЕСКАЯ ОРГАНИЗАЦИЯ БАЗ ДАННЫХ

6.1 Структуры внешней памяти, методы организации индексов

6.1.1 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 и управлением размещения наборов данных, включающий процесс организации хранения данных с определением формата хранимой записи и классификации записей.*

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

Соответственно возникают следующие разновидности объектов баз данных:

- ***таблицы*** — основные объекты базы данных, большей частью непосредственно видимые пользователям;
- ***последовательности*** — объекты БД, используемые для формирования уникальных числовых величин;
- ***индексы*** — управляющие структуры, создаваемые по инициативе разработчика (администратора) баз данных или верхнего уровня системы в целях повышения эффективности выполнения запросов и обычно автоматически поддерживаемые нижним уровнем системы;
- ***представления (views)*** — хранимые предложения SQL (запросы на выборку), которые можно запросить как таблицу;
- ***триггеры (triggers)*** — хранимые процедуры, запускаемые при выполнении определенных действий с таблицей;
- ***хранимая процедура*** — выполняемый объект, реализованный с помощью процедурного расширения SQL, которому можно передать аргументы и получить от него сформированные результаты;
- ***хранимая функция*** отличается от хранимой процедуры тем, что возвращаемым результатом выполнения функции является некоторое единичное значение;
- ***хранимые пакеты*** представляют собой совокупность процедур, переменных и функций, объединенных для выполнения некоторой задачи;
- ***журнальная информация***, поддерживаемая для удовлетворения потребности в надежном хранении данных;
- ***служебная информация***, поддерживаемая для удовлетворения внутренних потребностей нижнего уровня системы (например, информация о связях между таблицами).

Любая СУБД основывается на конкретном комплексном решении задач, связанных с организацией хранения и управления данными. Рассмотрим лишь некоторые варианты таких решений.

6.1.2 Хранение таблиц в базе данных

Существуют разные способы физического хранения отношений, и от того, как это хранение организовано, зависит быстродействие работы с базой данных. Изначально выделялось два типа 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.

При организации хранения отношения по столбцам единицей хранения является атрибут отношения. В этом случае в среднем тратится меньше памяти, необходимой для хранения отношения, так как исключаются дублирующие значения атрибутов. Однако при такой организации хранения необходимо наличие дополнительных надстроек, обеспечивающих связь разрозненно хранящихся значений атрибутов в единый кортеж отношения.

В настоящее время в каждой конкретной СУБД существует свой формат хранения таблиц и других объектов баз данных. 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 обычное покортежное хранение отношений. Dbf-файл организован таким образом, что в одном файле хранится одна таблица базы данных (отметим, что dbf-файл носит при этом название «база данных»). Структура dbf-файла представлена следующим образом [18]:

- файл базы данных состоит из записи заголовка и записей с данными;
- в записи заголовка определяется структура базы данных и содержится другая информация, относящаяся к базе данных;
- непосредственно записи с данными следуют за заголовком (байты располагаются последовательно) и включают в себя фактическое содержимое полей;
- длина записи (в байтах) определяется суммированием указанных длин всех полей;
- запись — это строка символов, состоящая из частей (полей) строго определенного размера. Эти размеры указаны в структуре описания.

В таблице 6.1 представлена структура dbf-файла.

Таблица 6.1 – Структура dbf-файла

Количество байт	**Наименование**
32	Заголовок DBF-файла
32	Описание первого поля
32	Описание второго поля
...	...
32	Описание n-го поля
1	Завершающий символ 0x0D (13)
RecordSize	Первая запись из n-полей

продолжение на следующей странице

Таблица 6.1 — Продолжение

Количество байт	Наименование
RecordSize	Вторая запись из n-полей
...	...
RecordSize	m-я запись из n-полей, где m=RecordsCount
1	Завершающий символ 0x1A (26)

RecordSize (размер записи в байтах) и RecordsCount (количество записей), значения которых берутся из заголовка DBF-файла.

В зависимости от СУБД физически все таблицы могут храниться в одном файле (MS Access, Mysql и др.) или для каждой таблицы может быть выделен отдельный файл (dBase). Кроме этого, также могут быть выделены файлы отдельных форматов для хранения других объектов БД.

6.1.3 Организация индексов, методы хранения и доступа к данным

Во всех существующих на рынке СУБД имеется в наличии средство, оптимизирующее дисковое пространство для хранения данных, а также обеспечивающее оптимальный по скорости доступ к данным. Такая надстройка над данными называется индексами.

*Как отмечалось в предыдущем разделе, **индекс** представляет собой некий упорядоченный указатель на записи в таблице.*

Понятие «указатель» означает, что индекс представляется как совокупность значений одного или нескольких полей таблицы БД и адресов страниц данных, где физически располагаются эти значения. То есть индекс состоит из пар значений «значение поля» — «физическое расположение этого поля». При этом индекс не является частью таблицы — это отдельный, взаимосвязанный с таблицей (или таблицами) объект БД. В целом индекс можно описать как специальную структуру данных, создаваемую автоматически или по запросу пользователя.

Пример 6.1

Индексы принято сравнивать с библиотечным каталогом, в котором информация о книгах записана на карточках и упорядочена по алфавиту или по темам, а в каждой карточке указано, где именно в хранилище располагается данная книга. Таким образом, библиотекарь по запросу читателя не просматривает весь библиотечный фонд, а берет книгу из конкретного места в книгохранилище, на которое указывает библиотечная карточка. То есть работа с индексом выглядит так же, как и с предметным указателем.

Поиск данных в таблице без использования индекса можно сравнить с последовательным перебором всех книг в библиотеке. Большинство таблиц в БД имеют большое количество записей, которые хранятся в определенном формате, и поиск необходимых данных по заданному критерию запроса путем последовательного перебора таблицы — запись за записью, естественно, может занимать большое количество времени. Индекс позволяет быстро искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов обеспечивается несколькими факторами, во-первых, за счёт того, что индекс имеет специальную структуру, оптимизированную под поиск, во-вторых, сами таблицы в БД могут храниться таким образом, чтобы обеспечивать оптимальный доступ к индексируемым полям.

Фактически, 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 записи таблицы по некоторому ключу.

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. Один вид организации индекса отличается от другого главным образом по способу поиска ключа с заданным значением [1].

Различают следующие методы хранения и доступа к данным: физический последовательный, индексно-последовательный, индексно-произвольный, инвертированный, метод хеширования и др. Рассмотрим некоторые из них, основываясь, в том числе, на сведениях из [9].

Инвертированный метод (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 рисунке 6.1 приведен пример поиска записей инвертированным методом доступа.

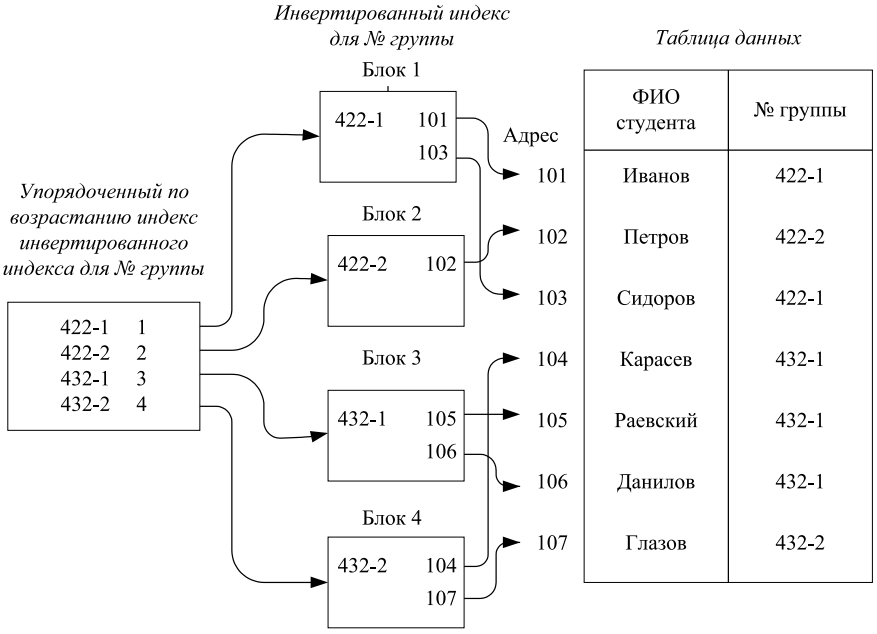


Рис. 6.1 – Инвертированный метод доступа

Прямой метод доступа

Основная особенность прямого метода доступа заключается во взаимно однозначном соответствии между ключом записи и ее физическим адресом. Физическое местоположение записи определяется непосредственно из значения ключа. 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. 6.2).

Рис. 6.2 – Пример цепочки синонимов

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 единица 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.

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 хеш-функции (со значением свертки большего размера).

Применительно к базам данных такие действия абсолютно неприемлемы. 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 хеширования:

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, но при этом возникает нарушение логической упорядоченности файла;

8) хеширование допускает возможность вторичного преобразования по специальному ключу.

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 [19]. Пример двоичного индекса представлен на рисунке 6.3.

Код студента	ФИО студента	Страна проживания	Дата рождения	№ группы
1	Карасев А.А.	Россия	27.08.95	412-1
2	Данилов О. В.	Казахстан	27.08.95	432-1
3	Раевский А. И.	Россия	20.05.95	432-1
4	Иванкова И.С.	Казахстан	11.10.96	421-1
5	Авдеев Н.В.	Кыргызстан	01.04.96	421-1
6	Антонова А.Н.	Украина	08.09.96	423-3
7	Караваев И.Н.	Россия	12.11.96	424-1
8	Красников И.И.	Беларусь	12.02.96	412-2

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. 6.3 – Таблица «Студент» и масочный индекс для таблицы «Студент»

В данном примере представлены двоичные маски для таблицы «Студент». Ключом индекса является поле «Страна проживания». В таблице «Студент» 8 записей. В маске каждый бит соответствует одной записи и устанавливается равным 1, если значение атрибута «Страна проживания» совпадает со значением параметра для этой маски. В таблице «Студент» в записях 1, 3, 7 указана страна проживания Россия, в строке 8 — Беларусь и т. д. Такой метод индексирования широко применяется в СУБД Oracle.

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 [19].

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 в совокупности 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.

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.

6.1.4 Словарь данных

Словарь данных *— это хранилище информации обо всех объектах, входящих в состав БД.*

СУБД использует словарь для получения информации об объектах и ограничения 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;
- значения по умолчанию для полей таблиц;
- объем распределенного и в данный момент времени используемого объектами в базе данных пространства;
- информацию аудита, например имена пользователей, обращавшихся к различным объектам и обновлявших данные.

Доступ к словарю данных возможен только в режиме чтения.

Одной из составляющих словаря данных и ключевым компонентом всей информационной структуры, являются ***системные таблицы*** БД, содержащие служебную информацию. 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 случаев 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 обеспечить выполнение следующих функций [9]:

- установление связи с другими пользователями;
- осуществление простого и эффективного управления элементами данных при вводе в систему новых элементов или изменения описания существующих;
- уменьшение избыточности и противоречивости данных;
- определение степени влияния изменений в элементах данных на всю БД;
- централизация управления элементами данных с целью упрощения проектирования БД и расширения ее структуры.

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

Такой словарь должен [9]:

1) поддерживать концептуальную, физическую и внешние модели данных;
2) быть интегрированным в СУБД;
3) поддерживать возможность хранения нескольких версий программной реализации;
4) обеспечивать эффективный обмен информацией между внешними программами и СУБД (в идеале привязка внешних и внутренних моделей должна происходить во время выполнения программ, использующих БД, при этом должно осуществляться динамическое построение описания БД).

6.1.5 Прочие объекты базы данных

Остановимся более подробно на других объектах БД, кратко представленных в 6.1.1.

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 на выборку, 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.

Фактически, любой запрос на выборку, представленный в разделе 5, Вы можете сохранить в базе данных и в дальнейшем вызывать как обычную таблицу. Единственное отличие в том, что результирующие наборы данных, получаемые при выполнении сложных запросов, обычно недоступны для редактирования.

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.

Одна из важных возможностей триггеров — это возможность реализации с их помощью функций аудита — когда в отдельной части базы данных фиксируется вся информация об активности пользователей в системе, например кто, когда и в каких таблицах изменял данные.

Хранимые пакеты, процедуры и функции находятся в ***словаре данных***. Там же сохраняется их исходный код.

Хранимая процедура *— это выполняемый объект, которому можно передать аргументы и получить 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 определяться относительно одной или нескольких таблиц БД [18].

Последовательности *— это объекты БД, введенные в некоторые СУБД (Oracle, MS Access 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 поддерживаются 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.

Во втором варианте поддерживается общий журнал изменений БД, 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 — поддержание только общего журнала изменений базы данных, который используется и при выполнении индивидуальных откатов.

Служебная информация

Как отмечалось выше, 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 БД, например числа столбцов таблицы, их размера, типов данных; описаний 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.

Например, в СУБД MS Access существуют служебные (системные) таблицы, в которых хранится, в том числе, информация обо всех объектах, созданных в базе данных. На рисунке 6.4 приведена структура системной таблицы MSysObjects БД СУБД MS Access.

MSysObjects

Имя поля	Тип данных
Connect	Поле MEMO
Database	Поле MEMO
DateCreate	Дата/время
DateUpdate	Дата/время
Flags	Числовой
ForeignName	Текстовый
Id	Числовой
Lv	Поле объекта OLE
LvExtra	Поле объекта OLE
LvModule	Поле объекта OLE
LvProp	Поле объекта OLE
Name	Текстовый
Owner	Двоичный
ParentId	Числовой
RmtInfoLong	Поле объекта OLE
RmtInfoShort	Двоичный
Type	Числовой

Рис. 6.4 – Структура таблицы MSysObjects БД СУБД MS Access

Еще раз заметим, что в большинстве систем управления базами данных системные таблицы и другие служебные данные скрыты от пользователей и недоступны для редактирования.

6.2 Оптимизация работы с базами данных

Можно дать некоторые рекомендации, которые позволят добиться повышения быстродействия и уберегут начинающих разработчиков баз данных от ошибок, которые могут возникнуть при проектировании и разработке баз данных.

1. Создавайте таблицы, не содержащие избыточных данных, — стремитесь к нормализации.
2. Создавайте индексы для сортируемых и объединяемых полей, а также для полей, используемых при задании критериев запроса в SQL-запросах. Повышение быстродействия при выполнении SQL-запросов можно достичь индексацией полей, являющихся внешними ключами.
3. Определяйте тип данных полей с учетом максимально точно подходящего типа 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.
4. При выборе типа данных, на котором определяется поле, следует учитывать:
   - 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, поддерживающих возможность работы с OLE-объектами.

5. Первичные и внешние ключи следует по возможности определять только на числовых полях либо на полях типа Дата/время, если это поле входит в составной первичный ключ.
6. Если первичный ключ может быть построен не менее чем на четырех полях — следует заменить его суррогатным ключом.
7. Таблицы справочники-классификаторы создавайте только для реально повторяющихся значений — например, нет смысла в базе данных создавать отдельные справочники-классификаторы для фамилий, имен и отчеств, достаточно в таблице с описанием людей выделить три отдельных поля: Фамилия, Имя, Отчество.
8. Необходимо периодически производить сжатие базы данных.

При наличии запоминающих устройств с большим объемом памяти проблема сжатия данных все же не утратила своей актуальности. Действительно, с приходом новых технологий появилась возможность создания БД с большим объемом хранимой в них информации (например, распределенные БД с таблицами, содержащими гигабайты данных), но для хранения таких БД по-прежнему приходится применять технологию сжатия данных.

Естественно, что механизм сжатия данных должен быть обратим. Преимущества СУБД, использующих сжатие данных [6]:

- 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 (работы с копией БД или объектами, допускающими синхронизацию данных) возможно использование обычных файловых архиваторов;
- для хранения сжатых данных при резервном копировании требуется меньше устройств резервного копирования;
- 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.

9. Следует удалять индексы, необходимость в которых отсутствует (см. раздел 6.1.3).
10. Используйте буферы оперативной памяти для временного хранения данных.

В настоящее время существуют СУБД, способные обрабатывать данные в оперативной памяти на качественно 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, поддерживаемых в оперативной памяти, их состояние сохраняется в некоторых контрольных точках в виде дисковых копий. Такие контрольные точки возникают в периоды наименьшей активности пользователей.

6.3 Экстенсиональная и интенсиональная части базы данных

Абстрагируясь от конкретных СУБД и внимательно анализируя реальное содержимое базы 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 таблицах (отношениях-каталогах) и содержит главным образом имена основных таблиц, имена и типы данных их атрибутов.

Во-вторых, это собственно наборы записей, содержащие пользовательские данные, сохраняемые в определенных разработчиками таблицах.

Наконец, в-третьих, это правила, определяющие ограничения целостности базы данных, триггеры и представления. 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 [1].

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 [1]; 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. Однако эти части не могут существовать по отдельности — они фактически дополняют друг друга.

Контрольные вопросы по главе 6

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?

Глава 7

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

7.1 СУБД первого поколения

Широкое распространение первые СУБД получили в 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 модели данных. Нельзя не отметить СУБД переходного типа, приближенные к реляционным, — системы с инвертированными файлами.

При создании первых СУБД внимание обращалось на способность систем хранить данные сложной структуры и значительного объема и использовать установленные связи между информационными элементами при проектировании приложений. Не менее важным достоинством являлась возможность обеспечения 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 (Ассемблер, КОБОЛ. PL/1), и эти 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 представлены системами IMS (иерархическая СУБД), IDS (сетевая СУБД), ADABAS (СУБД с инвертированными файлами) и соответствующими им отечественными СУБД ОКА, БАНК-ОС, ДИСОД.

Мы не будем подробно останавливаться на этих СУБД, поскольку в настоящее время они практически нигде не находят широкого применения. При необходимости с их описанием Вы сможете познакомиться в предыдущем издании данного учебного пособия [20].

7.2 СУБД второго поколения — реляционные СУБД

7.2.1 Архитектура СУБД второго поколения

В начале 80-х годов с началом широкого распространения персональных компьютеров получают широкое распространение СУБД, позволяющие оперировать данными, представленными в виде реляционной модели, и использующие языки манипулирования данными SQL, QBE. К классу реляционных систем управления базами данных относятся следующие СУБД:

- FoxPro (другие реализации систем на базе СУБД dBase), разработанная фирмой Fox Technologies и впоследствии купленная Microsoft;
- Access — разработка корпорации Microsoft;
- Oracle — разработка корпорации Oracle;
- MS SQL Server — разработка корпорации Microsoft;
- MySQL — свободно-распространяемая СУБД, разработку которой осуществляет корпорация Oracle.

Этот список можно продолжить, поскольку на рынке СУБД в настоящее время царит изобилие. Несмотря на это, идеологии реляционных СУБД имеют 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 отношений.

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 к серверу БД, на сервере средствами СУБД этот запрос обрабатывается, результаты обработки запроса передаются клиенту (рис. 7.1).

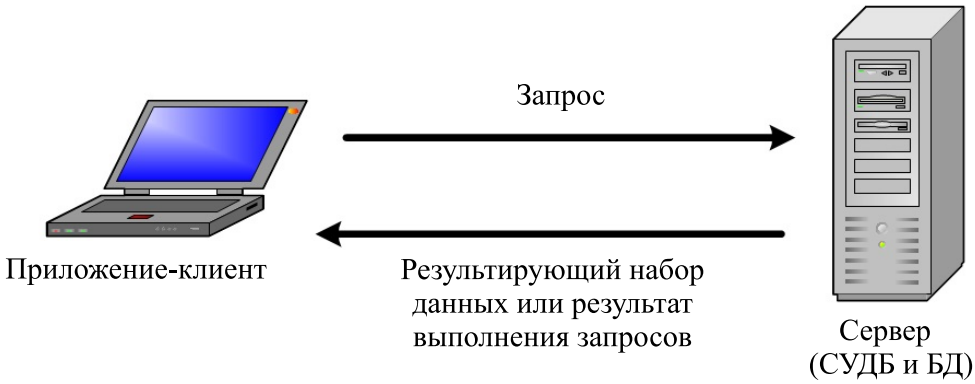


Рис. 7.1 – Принцип организации взаимодействия в архитектуре клиент-сервер

Целесообразно поручать мощному серверу выполнять крупные задачи, а клиентскому компьютеру — простые 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 изображениями. База данных, построенная с учетом сети и совместной обработки, называется распределенной БД.

На рисунке 7.2 изображен пример двухзвенной клиент-серверной архитектуры. На практике могут использоваться и многозвенные конструкции, например отдельно может выделяться сервер для функционирования СУБД и БД, отдельно сервер приложений, на котором устанавливается программа, отдельно клиентская часть.

При файл-серверном подходе обработка запроса к БД происходит непосредственно на компьютере средствами СУБД, установленной на этом компьютере, и с которого этот запрос был послан. При этом сама база данных может находиться удаленно на выделенном файл-сервере. Фактически, для обработки запроса СУБД переносит все необходимые данные с сервера на компьютер клиента (рис. 7.2), что естественно снижает быстродействие при работе с большими объемами данных.

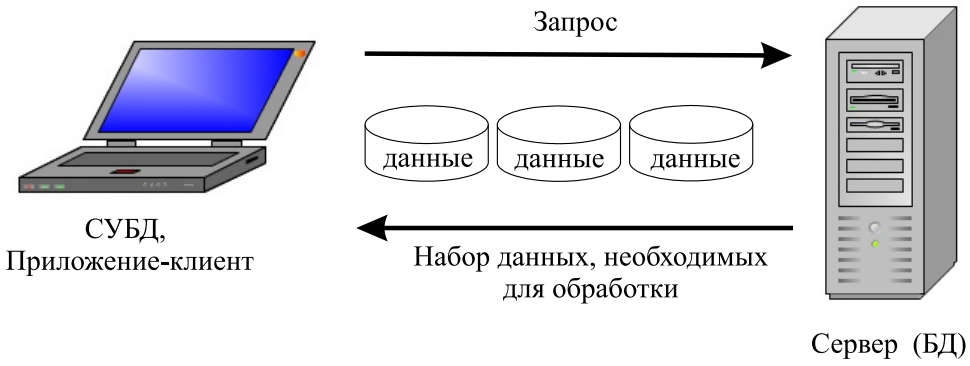


Рис. 7.2 – Принцип организации взаимодействия в архитектуре файл-сервер

СУБД FoxPro, MS Access являются представителями класса файл-серверных СУБД. Однако в ряде случаев с помощью, например, СУБД MS Access можно создавать клиент-серверные приложения, используя в качестве серверов БД какие-либо другие СУБД (Oracle, MS SQLServer и др.).

7.2.2 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.

Последняя версия оригинальной СУБД FoxPro — версия 2.6 — работала под Mac OS, DOS, Windows и Unix. В 1992 году Fox Technologies объединилась с Microsoft, новые версии продукта обрели ряд новых функций и приставку «Visual». В версии Visual FoxPro 3.0 список поддерживаемых платформ сократился до Mac OS и Windows, а в более поздних версиях — уже только до Windows.

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 [18]:

- 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. Число 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 мемо-полями.

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. 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).

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.

Visual FoxPro (VFP) — визуальная среда разработки пользовательских приложений и БД, выпускаемая корпорацией Microsoft. Последней версией является версия 9.0. В этой среде применяется язык программирования FoxPro. Среда разработки версии 7.0 может работать в операционных системах Windows 9x и ядра NT и старше, версии 8.0 и 9.0 — только в Windows XP и старше.

Разработка продукта прекращена с выходом SP2 для версии 9.0, поддержка продукта осуществляется только до 2015 года.

7.2.3 СУБД MS Access

Первая версия СУБД MS Access была разработана фирмой Microsoft в 1992 году и не получила широкого распространения из-за отсутствия стабильных встроенных средств разработки и ведения баз данных. Первой из версий MS Access, получившей признание разработчиков, явилась СУБД MS Access 2.0, которая, как и последующие версии, входит в состав соответствующих версий MS Office. СУБД MS Access позволяет управлять всеми сведениями из одного файла базы данных или разделять базу данных на несколько файлов, сохраняя, например, в одном файле все таблицы, а в другом — все остальные объекты БД.

В рамках файла базы данных 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 (до версии MS Access 2003 включительно).

Внешний вид окна MS Access, с созданными ранее таблицами, представлен на рисунке 7.3.

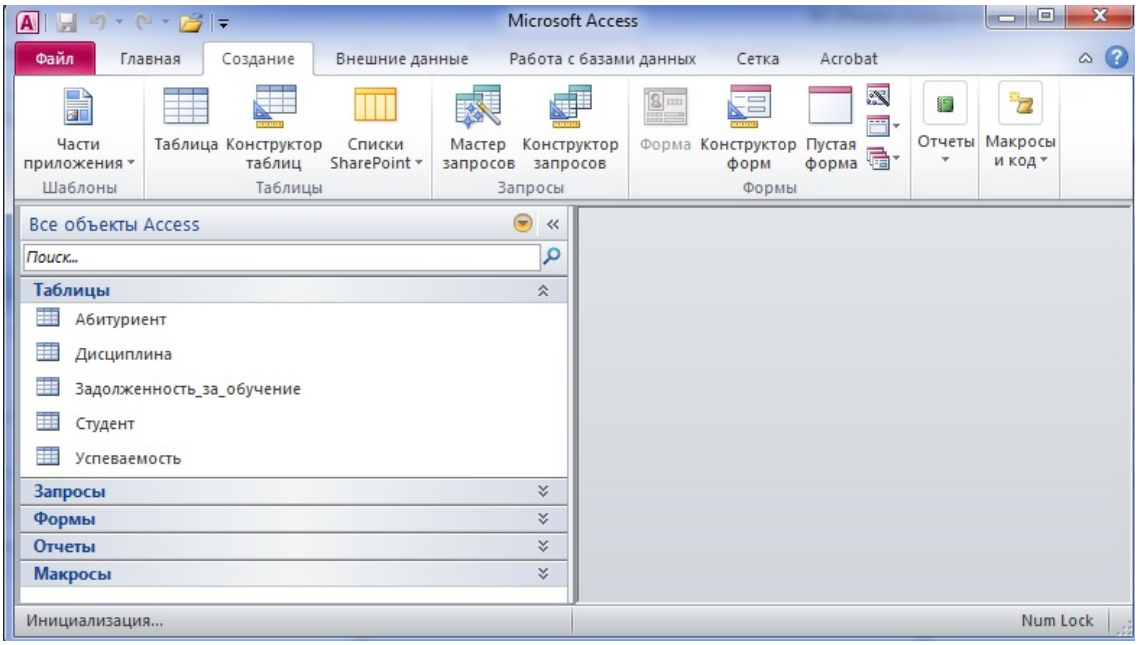


Рис. 7.3 – Окно БД MS Access

Ниже представлены характеристики БД в СУБД MS Access, взятые из файла описания СУБД:

- 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) — 1000;
- 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 создаются формы. Формы являются типом объектов базы данных, обычно используемым 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 выводит 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 веб-страниц, 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 использо-

вать за пределами MS Access, предоставляя пользователям возможность обновлять или просматривать данные через Интернет или интрасеть, для чего пользователям требуется Internet Explorer или другой браузер. 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, например, как нажатие кнопки. Начиная с Access 97, 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 содержатся 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 жизни. Значение любой переменной или константы, определенной или существующей 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, можно также связать таблицу саму с собой. Связи «многие-ко-многим» (M:M) возможно реализовать только через промежуточные таблицы. Для связей, в которых проверяется целостность данных, разработчик 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, имеется возможность импорта и экспорта таблиц в другие БД, а также возможность слияния таблиц и отчетов с другими продуктами, входящими в состав MS Office.

7.3 СУБД третьего поколения и объектно-ориентированные СУБД

7.3.1 Манифесты СУБД третьего поколения и объектно-ориентированных СУБД

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 систем управления базами данных (ОО-СУБД).

Еще в 1990 году Комитетом по развитию функциональных возможностей СУБД был опубликован доклад, представленный на конференции в Виндермере (Англия) и получивший название «Манифест СУБД третьего поколения». Полный текст документа изложен в [21], в данном пособии рассмотрим лишь общие положения Манифеста.

Авторы документа отмечают, что на момент опубликования доклада 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, явившиеся важным шагом в эволюции, с которым связаны использование непроцедурного языка манипулирования данными и обеспечение в существенной степени независимости данных.

Авторы манифеста сформулировали три принципа и тринадцать предложений, имеющих отношение к СУБД третьего поколения.

Принципы:

1) СУБД третьего поколения будут обеспечивать сервисы в трех областях:
   - управления данными;
   - управления объектами;
   - управления знаниями;
2) СУБД третьего поколения должны сохранить функции непроцедурного доступа и независимости данных, присущие СУБД второго поколения;
3) СУБД третьего 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, средствами для выполнения удаленных операций над данными, а также распределенными возможностями.

Предложения были условно разделены на три группы.

Первая группа содержит предложения, связанные с ОО-свойствами, которые необходимы для систем третьего поколения:

1) наличие объектно-ориентированных возможностей, но не как элементов полностью нового архитектурного рассмотрения СУБД, а как расширение существующих моделей;
2) поддержание механизма наследования;
3) поддержание функций (методов, процедур) и инкапсуляции;
4) факультативное назначение уникальных идентификаторов 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 является языком 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 должны быть физическими, а не логическими и не иметь ничего общего с моделью данных.

Третья группа предложений касается идеологии открытых систем:

1) СУБД третьего поколения должны быть многоязычными;
2) необходимо близкое соответствие между типами данных СУБД третьего поколения и используемыми языками программирования, а также должны быть исключены несоответствия различных стандартов описания языка SQL;
3) SQL должен быть сохранен в системах третьего поколения, несмотря на многочисленные недостатки;
4) SQL-запросы и ответы на них должны быть самыми нижними уровнями коммуникаций между клиентом и сервером.

После опубликования «Манифеста СУБД третьего поколения» эти принципы и предложения стали играть важную роль в мире реляционных БД, расширенных объектно-ориентированными возможностями.

За год до опубликования «Манифеста СУБД третьего поколения» другая группа ученых в области БД опубликовала документ под названием «Манифест объектно-ориентированных систем баз данных», и если в «Манифесте СУБД третьего поколения» сформулированы перспективные направления в разработке так называемой гибридной модели (т. е. смеси реляционной и объектно-ориентированной), то в Манифесте ОО-систем сделано то же самое для сугубо объектно-ориентированных БД (ООБД). Этот документ содержит тринадцать «заповедей» относительно обязательных свойств ООБД [22].

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 (представляют собой один и тот же объект) или они могут быть равны (имеют одно и то же значение). Это влечет два следствия: разделяемость и изменяемость объектов.

3. ***Инкапсуляция объектов***.

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. ***Поддержка типов и классов***.

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. ***Обеспечение 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. Для обеспечения перекрытия и перегрузки система не должна связывать имена операций с программами во время компиляции. Поэтому имена операций должны разрешаться (транслироваться в адреса программ) во время выполнения. Отложенная трансляция называется *поздним связыванием*.

7. ***Обеспечение вычислительной полноты***.

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 неспособна 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 БД поставляется с набором предопределенных типов. Эти типы могут при желании использоваться программистами для написания приложений. Набор предопределенных типов должен быть расширяемым в следующем смысле: необходимо иметь средства для определения новых типов и не должно быть различий в использовании системных и определенных пользователем типов. Конечно, способы поддержания СУБД системных и пользовательских типов могут значительно различаться, но эти различия должны быть невидимыми 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 нет необходимости в том, 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 процесса с целью последующего использования в другом процессе. Свойство стабильности должно быть ортогональным: для каждого объекта вне зависимости от его типа должна иметься возможность сделать его стабильным (т. е. без какой-либо явной переделки объекта); кроме того, неявным: пользователь не должен явно перемещать или копировать данные, чтобы сделать их стабильными.

10. ***Управление вторичной памятью***.

Управление вторичной памятью является классической чертой систем управления базами данных. Эта возможность позволяет управлять большими базами данных и обычно поддерживается с помощью набора механизмов управления индексами, кластеризации данных, буферизации данных, выбора пути доступа и оптимизации 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. По крайней мере, должна 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:

1) 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»;
2) быть эффективным, т. е. формулировка запросов должна допускать возможность оптимизации запросов;
3) 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 объектно-ориентированных СУБД:

- 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.

Рассмотрим два похода к новым типам СУБД, предложенные в манифестах.

Оба документа предполагают поддержку сложных объектов, инкапсуляции, структуры классов, наследования и расширяемости [21, 22]. Однако использоваться эти механизмы должны по-разному. Наиболее значительные расхождения обнаруживаются в области использования таких механизмов, как уникальные идентификаторы объектов, поддержка полиморфизма и языковые средства. Здесь позиции авторов манифестов 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.

Сегодня, по проишествии более двух десятков лет с момента публикации этих документов можно сказать, что ряд положений обоих манифестов реализован в современных СУБД, о чем будет рассказано ниже, однако при этом отказаться полностью от реляционной модели данных не всегда возможно.

7.3.2 Общие понятия объектно-ориентированного подхода к базам данных

Одним из аргументов в пользу внедрения объектно-ориентированных возможностей в системы управления базами данных является наличие более мощного и более высокоуровневого семантического моделирования по сравнению с системами, в основе которых лежит чистая реляционная модель данных. Базовую основу объектно-ориентированного подхода составляют следующие понятия:

- объект и идентификатор объекта (индивидуальность объекта);
- атрибуты и методы;
- классы;
- иерархии и наследование 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 ООСУБД, 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 выполнений 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, 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 проблемы 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 располагает разнообразными средствами манипуляции с объектами базы данных.

7.3.3 Реализация объектно-ориентированного подхода в СУБД Oracle

СУБД Oracle, начиная с версии 8i, предоставляет расширенный набор встроенных типов данных, а также позволяет конструировать новые типы данных со спецификацией методов доступа к ним, т. е. фактически это означает наличие инструмента, позволяющего строить структурированные типы данных, непосредственно отображающие сущности предметной области. В данном разделе будем опираться на материалы, изложенные в [19, 23, 24].

Oracle 8i 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 версии 12c 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++.

Объектные типы

В последних версиях Oracle определяемые пользователем типы данных могут применяться в качестве следующих составляющих:

- 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 7.1

```
CREATE type employee_type as object (
ssn number (9),
name varchar2 (35),
address varcher2 (70),
resume varchar2 (250));
```

На рисунке 7.4 представлено окно в среде PL/SQL Developer (среда для работы с БД Oracle), в котором отображается созданный объектный тип.

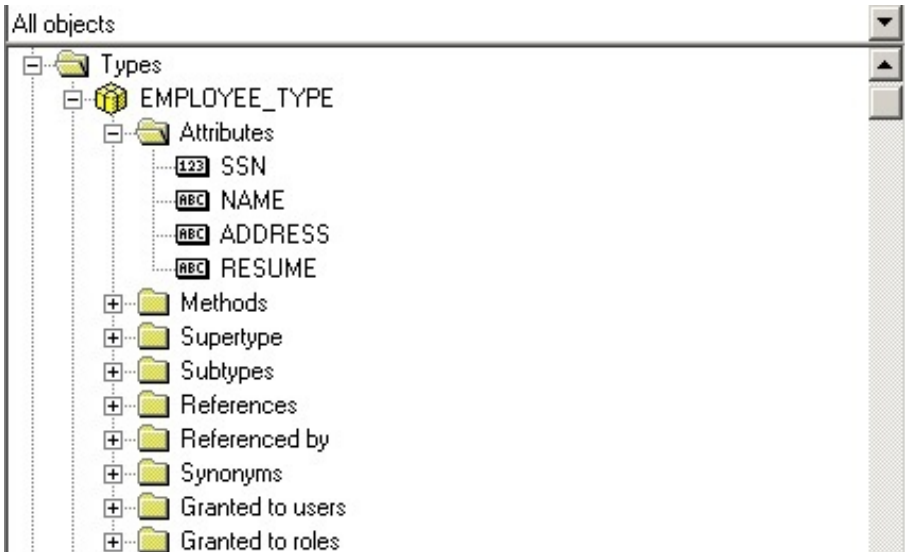


Рис. 7.4 – Отображение пользовательского объектного типа Employee_type в БД Oracle

Пример 7.2

```
CREATE table Manager_candidates (
Position varchar2 (40),
Vacancy varchar2 (250),
Employee employee_type);
```

На рисунке 7.5 представлена структура созданной таблицы «Manager_candidates», в которой атрибут «Employee» определен на объектном типе данных «Employee_type».

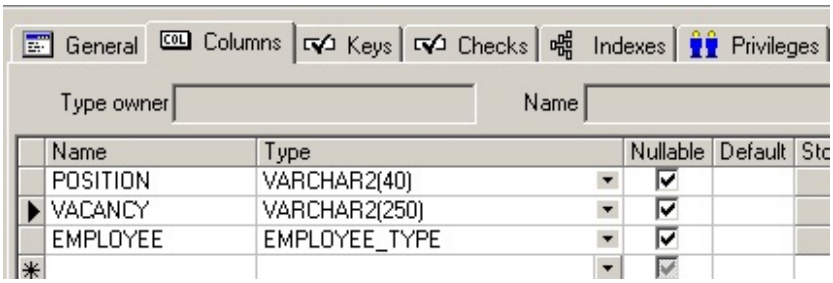


Рис. 7.5 – Структура таблицы «Manager_candidates» с атрибутом Employee

```
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;
```

В результате выполнения этого запроса на выборку будет выдано значение атрибута «Employee.name» из таблицы «Manager_candidates» (рис. 7.6).

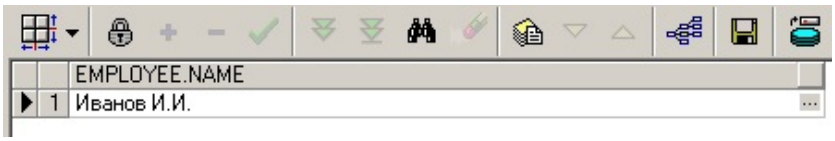
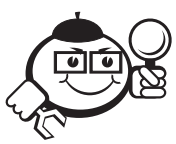


Рис. 7.6 – Результат выполнения запроса на выборку (Пример 7.2)



Пример 7.3

```
CREATE table employee of employee_type;
```

Следует отметить, что для извлечения значения атрибута объекта внутри таблицы необходимо использовать псевдоним и точечную нотацию (пример 7.2).

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 7.4

```
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»: 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 (*storage 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, убеждены, 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 9i интегрирована возможность и функциональность Express Server. 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 в единой базе данных Oracle, а для работы с многомерными кубами используются JavaBeans, входящие в состав инструментария.

Поддержка XML в Oracle

Сервер Oracle поддерживает не только реляционную, объектную, многомерную модели данных, но и XML (eXtensible Markup Language — расширяемый язык разметки). Начиная с версии 9i 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 БД Oracle в настоящее время 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 найти 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, одновременно работающий с реляционными данными, очередями сообщений, XML-данными, пространственными данными, контекстом. И наоборот, создав над реляционными или объектными таблицами базы данных представление XMLType View, можно работать с этими данными через XML-интерфейс. В Oracle поддерживаются стандарты доступа SQLX, Xpath, DOM, Java-Beans и JNDI.

СУБД Oracle обеспечивает возможность оперировать с XML-документами так же, как и с реляционными данными. Подсистема, названная Oracle XML DB, вошла в состав второго релиза Oracle 9i. Oracle XML DB дает пользователям возможность наряду с реляционными данными применять XML-документы, не преобразуя их в строки и колонки. XML-документы хранятся как объекты Oracle.

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, которую осуществить невозможно.

Пример 7.5

```
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 представления в реляционных БД, но в них могут использоваться строгая типизация, сложные структуры (не в первой нормальной форме), методы и возможности ссылок на существующие реляционные данные.

В заключение 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.

Также заметим, что с каждой новой версией разработчики Oracle вносят все новые и новые объектно-ориентированные возможности и совершенствуют уже существующие.

7.3.4 СУБД Caché

СУБД Oracle смело можно отнести к классу гибридных объектно-реляционных СУБД, поскольку здесь успешно сочетаются реляционный и объектно-ориентированный подход. В этом разделе мы познакомимся с СУБД принципиально нового постреляционного класса. При написании данного раздела были использованы материалы, опубликованные в [25, 26].

При всех достоинствах современной объектной технологии разработки БД 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 Caché компании 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 роли шлюза к реляционным базам данных.

Отличительной особенностью СУБД Caché является независимость хранения данных от способа их представления, что реализуется с помощью так называемой единой архитектуры данных Caché.

В рамках данной архитектуры существует единое описание объектов и таблиц, отображаемых непосредственно в многомерные структуры ядра БД, ориентированного на обработку транзакций. Как только определяется класс объектов, Caché автоматически генерирует реляционное представление данных этого класса. Подобным же образом, как только в словарь данных поступает DDL-описание на языке SQL, Caché 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 Caché 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 Caché 4.0. реализована технология создания динамических web-приложений CSP (Caché Server Pages). Наряду с этим системная библиотека Net предоставляет классы, реализующие протоколы SMTP, POP3, HTTP, FTP и др. На рисунке 7.7 представлена архитектура СУБД Caché.

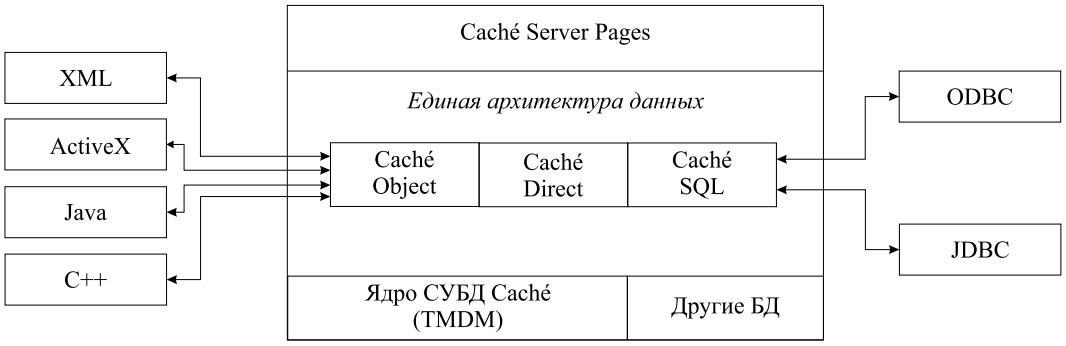


Рис. 7.7 – Архитектура системы Caché

Основные компоненты СУБД Caché

TMDM.

TMDM *— многомерное ядро системы, ориентированное на работу с транзакциями.*

Данные в Caché 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 Caché 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 Caché реализованы атомарные операции добавления и удаления без проведения блокировки, в частности это применяется для счетчика идентификаторов объектов.

Сервер Caché Objects. Сервер обеспечивает представление многомерных структур данных ядра системы в виде объектов, инкапсулирующих как данные, так и методы их обработки.

Объектная модель Caché соответствует объектной модели стандарта ODMG (*Object Data Management Group*). В соответствии со стандартом ODMG каждый экземпляр объекта в Caché 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 Caché представлена на рисунке 7.8. В соответствии со стандартом в Caché реализовано два типа классов:

1) классы типов данных (литералы);
2) классы объектов (объекты).

Классы типов данных определяют допустимые значения констант (литералов) и позволяют их контролировать. Литерал не имеет дополнительной идентифика-

ции, кроме своего значения, в то время как объекты имеют еще и уникальную идентификацию.

Классы типов данных подразделяются на два подкласса типов:

1) атомарные;
2) структурированные.

Атомарными литеральными типами в Caché являются традиционные скалярные типы данных (String, Integer, Float, Date и др.). В Caché реализованы две структуры классов типов данных — список и массив. Каждый литерал уникально идентифицируется индексом в массиве или порядковым номером в списке.

Рис. 7.8 – Объектная модель Caché

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 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 Caché в полном объеме поддерживает все основные концепции объектной технологии:

- наследование. Объектная модель Caché позволяет наследовать классы от произвольного количества родительских классов;
- полиморфизм. Объектная модель Caché позволяет создавать приложения целиком и полностью независимыми от внутренней реализации методов объекта;

- инкапсуляцию. Объектная модель Caché 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 относится к интерфейсной части;
- хранимость. Система Caché поддерживает несколько видов хранения объектов: автоматическое хранение в многомерной базе данных Caché; хранение в любых структурах, определенных пользователем; хранение в таблицах внешних реляционных баз данных, доступных через шлюз Caché SQL Gateway.

Сервер Caché SQL. Сервер осуществляет представление многомерных структур данных в виде реляционных таблиц. Наряду с реализацией в полном объеме основных принципов объектной технологии в СУБД Caché 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 Caché возможно автоматическое преобразование описаний реляционных таблиц в классы объектов. При поступлении на сервер Caché SQL DDL-описания реляционной таблицы Caché 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 Caché. Это позволяет Caché работать с данными как в виде реляционных таблиц, так и в виде классов объектов. Таким образом, при переходе с реляционной технологии на объектную разработка не начинается с «нуля» — многое делается Caché автоматически.

Сервер прямого доступа (Caché Direct). Сервер обеспечивает предоставление прямого доступа к многомерным структурам данных ядра системы. С помощью сервера Caché Direct разработчик получает доступ к многомерным структурам ядра системы. В СУБД Caché реализован собственный язык программирования COS (*Caché Object Script*). 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 Caché вводится понятие Caché-программы, которая не является составной частью классов и предназначена для написания прикладного программного обеспечения для текстовых терминальных систем. COS предоставляет ряд функций для работы с массивами данных, или глобалями, составляющими ядро системы.

Использование прямого доступа к данным позволяет оптимизировать время доступа к ним. Для прямого доступа и работы с многомерными структурами ядра системы можно воспользоваться утилитой эмуляции ASCII-терминала Caché Terminal, которая обычно используется для целей обучения языку COS и тестирования работы терминального приложения.

В стандартной поставке системы разработчику предлагается два средства администрирования Caché: *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 баз данных Caché — 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 Caché 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 Caché. В Caché реализован собственный протокол для работы БД в распределенной среде, носящий название DCP *(Distributed Caché Protocol*). С помощью интерфейсов Configuration Manager можно определить источники данных, а также связи между различными компонентами сети;
- настроить систему Caché. Разработчику предоставляется возможность конфигурирования различных компонентов Caché, таких как параметры журналирования, настройки теневых серверов, параметры сервера лицензий, параметры Caché-процессов и другие.

Утилита *Control Panel* предоставляет схожий набор функций администрирования, но также позволяет осуществлять:

- управление процессами Caché;
- настройку 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 Caché выполняется на уровне глобалей;
- определение теневых серверов системы;
- создание резервных копий баз данных.

Основой концепции серверных страниц Caché является автоматическое создание по запросу пользователя web-страниц, содержащих требуемую информацию из БД Caché. Вся бизнес-логика CSP-приложений выполняется в непосредственной близости к хранилищу данных Caché, таким образом сокращается объем данных, которыми обмениваются web-сервер и сервер БД Caché, 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 Caché и web-сервером используются высокоскоростные API-интерфейсы.

Серверные страницы Caché представляют собой HTML-файлы, содержащие дополнительные теги Caché (Caché Application Tags или CATs). Для создания CSP-приложений можно воспользоваться стандартными средствами разработки HTML-страниц (Caché предоставляет add-in модуль для полной интеграции с Macromedia DreamWeaver) или, на крайний случай, обыкновенным текстовым редактором.

СУБД Caché 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 Caché автоматически определяет региональные настройки операционной системы и устанавливает соответствующую схему локализации. Также предоставляется возможность установки Unicode-версии (16bit) Caché.

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, Caché предоставляет ODBC- и JDBC-драйверы для представления данных из СУБД Caché в виде реляционных таблиц и работы с ними.

СУБД Caché предоставляет стандартные ActiveX-компоненты, которыми можно воспользоваться при создании пользовательского приложения в таких средствах разработки, как Visual Basic. Кроме этого, предоставляется мастер создания форм Caché Form Wizard для облегчения разработки пользовательских форм в среде Visual Basic.

Кроме всего перечисленного, в следующей версии Caché планируется обеспечить поддержку XML — общепринятого стандарта для обмена данными между различными платформами и SOAP-протокола для удаленного вызова функций.

7.3.5 Перспективы развития СУБД

Рассмотрев 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 взаимодействии с объектными типами, наследование и инкапсуляция будут использоваться на должном уровне;
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.

Подводя итог, можно сказать, что на сегодняшний день отказаться полностью от реляционной модели данных у разработчиков современных СУБД вряд ли получится. Однако внедрение объектно-ориентированных возможностей и их интеграция с реляционной моделью данных позволят обеспечить хранение и обработку больших объемов данных и их использование в автоматизированных информационных системах различных уровней сложности.

Контрольные вопросы по главе 7

1. Перечислите и охарактеризуйте СУБД первого и второго 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 Caché.
5. Опишите основные различия Манифестов ООБД и СУБД третьего поколения.
6. Охарактеризуйте общие понятия объектно-ориентированного подхода к БД.
7. Перечислите основные объектно-ориентированные возможности СУБД Oracle.

ЗАКЛЮЧЕНИЕ

Мы рассмотрели основы организации баз данных. Естественно, полноценно освоить представленный материал можно только на практике. В этом поможет руководство к выполнению лабораторных работ, где подробно рассказано об основах работы в среде проектирования PowerDesigner и в СУБД MS Access.

Приступая к разработке баз данных, необходимо помнить, что база данных не создается «с нуля» — в основе ее создания лежит изучение той предметной области, которую Вы собираетесь автоматизировать. Чем полнее Вы изучите предметную область, выделите основные объекты и принципы взаимодействия между ними, тем более качественно Вы сможете спроектировать модель данных этой предметной области, а на ее основе, собственно, и создать базу данных. Помните, что от качества разработанной базы данных во многом зависит качество создаваемой информационной системы.

Надеемся, что настоящее пособие будет востребовано студентами вузов и специалистами, желающими изучить основы организации баз данных. Полученные в ходе изучения данной дисциплины знания могут быть востребованы при изучении студентами направлений подготовки бакалавров «Программная инженерия», «Бизнес-информатика», «Государственное и муниципальное управление» таких дисциплин, как «Базы данных», «Организация баз данных», «Распределенные информационные системы» и других смежных дисциплин.

ЛИТЕРАТУРА

[1] Кузнецов С. Д. Основы баз данных / С. Д. Кузнецов. — 2-е изд. — М. : Интернет-Университет Информационных Технологий ; БИНОМ. Лаборатория знаний, 2007. — 484 с.

[2] Чудинов И. Л. Базы данных : учеб. пособие / И. Л. Чудинов, В. В. Осипова ; Томский политехнический университет. — Томск : Изд-во Томского политехнического университета, 2011. — 140 с.

[3] Матрин Дж. Организация баз данных в вычислительных системах / Дж. Матрин. — М. : Мир, 1980.

[4] U. Dayal et al. Third Generation TP Monitors: A Database Challenge — Proceeding of the 1993 ACM SIGMOD. — 394 p.

[5] Melton J. Understanding the New SQL: A Complete Guide / J. Melton, A. Simon. — San Francisco : Morgan Kaufmann Publishers, 1992 39–40.

[6] Саймон А. Р. Стратегические технологии баз данных: менеджмент на 2000 год / А. Р. Саймон. — М. : Финансы и статистика, 1999. — 479 с.

[7] E. F. Codd. A Relational Model of Data for Large Shared Data Banks // Communications of the ACM. — 1970, June.

[8] Кодд Э. Ф. Реляционная модель данных для больших, совместно используемых банков данных // Системы управления базами данных. — 1995. — N 1 — С. 145–160.

[9] Атре Ш. Структурный подход к организации баз данных / Ш. Атре. — М. : Финансы и статистика, 1983.

[10] Дейт К. Дж. Введение в системы баз данных // Introduction to Database Systems. — 8-е изд. — М. : Вильямс, 2006. — С. 1328.

[11] Пуле Мишель. Четыре грани целостности [Электронный ресурс]. — URL : http://www.osp.ru/win2000/sql/2000/01/010.htm (дата обращения: 20.05.2015).

[12] Вендров А. М. Проектирование программного обеспечения экономических информационных систем / А. М. Вендров. — М. : Финансы и статистика, 2006. — 544 с.

[13] CASE-Технологии [Электронный ресурс]. — URL : http://case-tech.h1.ru (дата обращения: 20.05.2015).

[14] Стратегическое управление информационными системами / Г. Калянов, [и др.] — М. : БИНОМ. Лаборатория знаний, 2014. — 510 с.

[15] Семизельникова О. А. Исследование возможностей CASE-технологии при создании интеллектуальных систем [Электронный ресурс]. — URL : http://www.inftech.webservis.ru (дата обращения: 20.05.2015).

[16] ISO/IEC 9075–11:2011 Information technology — Database languages — SQL [Электронный ресурс]. — URL: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=53682 (дата обращения: 20.05.2015).

[17] Ульман Дж. Основы систем баз данных / Дж. Ульман. — М. : Финансы и статистика, 1983.

[18] Попов А. А. Программирование в среде FoxPro 2.0. Построение систем обработки данных / А. А. Попов. — М. : Радио и связь, 1994. — 352 с.

[19] Использование Oracle 8/8i / Вильям Дж. Пейдж [и др.]. — М. : Вильямс, 2000. — 1024 с.

[20] Сенченко П. В. Организация баз данных : учеб. пособие / П. В. Сенченко. — Томск : Изд-во Томск. гос. ун-та систем управления и радиоэлектроники, 2005. — 203 с.

[21] Системы баз данных третьего поколения: Манифест / Комитет по развитию функциональных возможностей СУБД // Системы управления базами данных. — 1995. — N 2.

[22] Манифест систем объектно-ориентированных баз данных / М. Аткинсон [и др.] // Системы управления базами данных. — 1995. — N 4.

[23] Ривкин Марк. Новые возможности Oracle 9.2 // Открытые системы. — 2002. — N 11.

[24] Брила Б. Oracle 11g. Настольная книга администратора баз данных / Б. Брила, К. Луни — М. : Лори, 2012 — 864 с.

[25] СУБД Caché. Объектно-ориентированная разработка приложений / В. Кирстен [и др.]. — Питер, 2001.

[26] Сиротюк О. Постреляционная СУБД Cacheé [Электронный ресурс]. — URL : http://www.ict.edu.ru/ft/003849/cache.pdf (дата обращения: 20.05.2015).

СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

1:1 — связь «один-к-одному»

1:M — связь «один-ко-многим»

ANSI — American National Standards Institute

BCNF — нормальная форма Бойса—Кодда

CASE — Computer Aided Software Engineering

COS — Caché Object Script

CSP — Caché Server Pages

DB — DataBase

DBMS — DataBase Management System

DCL — Data Control Language

DCP — Distributed Caché Protocol

DML — Data Manipulation Language

ER — Entity-Relationship

ERX — Entity-Relationship eXpert

ISO — Международная организация по стандартизации

M:N — связь «многие-ко-многим»

NF (НФ) — нормальная форма

NIST — Национальный институт стандартов и технологий

OLAP — Online Analytical Processing

PL/SQL — Procedural Language/SQL

QBE — Query-by-Example

RDM — Relational Data Modeler

SDL — Schema Definition Language

SEQUEL — Structured English Query Language

SQL — Structured Query Language

SQL/PSM — SQL/Persistent Stored Modules

SOAP — Simple Object Access Protocol

WFF — Well-Formed Formula

XML — eXtensible Markup Language

БД — база данных

ВУЗ — высшее учебное заведение

ЖЦ — жизненный цикл

ОЗУ — оперативное запоминающее устройство

ОО — объектно-ориентированный

ПО — программное обеспечение

ПрО — предметная область

СУБД — система управления базами данных

ФИО — фамилия, имя, отчество

ЭВМ — электронная вычислительная машина

ГЛОССАРИЙ

Атрибут сущности — это именованная характеристика моделируемого сущностью объекта, являющаяся некоторым свойством сущности.

База данных — совокупность 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.

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

Внешний ключ — это атрибут (или совокупность атрибутов), значения которого однозначно характеризуют сущности, представленные кортежами другого отношения, т. е. соответствуют значению его первичного ключа.

Домен — множество допустимых значений атрибута определенного типа.

Журнал изменений БД — это часть БД, в которую поступает информация обо всех изменениях базы данных.

Индекс — упорядоченный указатель на записи в таблице.

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.

Объектный тип — это расширение типа, определяемого пользователем, позволяющее инкапсулировать методы с элементами данных в едином логическом модуле.

Операционная система — это базовый комплекс компьютерных программ, обеспечивающий управление аппаратными средствами компьютера, работу с файлами, ввод и вывод информации, а также выполнение прикладных программ и утилит.

Отношение есть множество кортежей, соответствующих одной схеме отношения.

Первичный ключ — минимально допустимый набор атрибутов, значения которых однозначно определяют кортеж отношения.

Поведение объекта — набор методов (программный код), оперирующих над состоянием объекта.

Связь — это графическая ассоциация между сущностями.

СУБД — это набор специальных программных приложений, предназначенных для обеспечения эффективного доступа к базе данных, используемый для предоставления только необходимой информации, обеспечения независимости от возможных изменений в структуре той части базы данных, которую не обрабатывает программа.

Сущность — это объект предметной области, элемент информационной системы или, другими словами, — класс однотипных объектов, информация о которых должна быть учтена в модели.

Транзакция — это последовательность операций над БД, рассматриваемых СУБД как единое целое.

Файл — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные.

Физическое представление — размещение физической структуры и значений хранимых данных в памяти компьютера (внешней и оперативной).

Учебное издание

Сенченко Павел Васильевич
ОРГАНИЗАЦИЯ БАЗ ДАННЫХ

Учебное пособие

Корректор Осипова Е. А.
Компьютерная верстка Морозова Ю. В.

Издано в Томском государственном университете
систем управления и радиоэлектроники.
634050, г. Томск, пр. Ленина, 40
Тел. (3822) 533018.