Часть 2.
1. БАЗЫ ДАННЫХ И СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Современные информационные системы (ИС), основанные на концепции банков данных и баз знаний, характеризуются большими объемами хранимой информации, сложной организацией, необходимостью удовлетворять разнообразные требования пользователей. Важным компонентом этой концепции является единая методология проектирования БД. БД, являясь информационной моделью непрерывно меняющегося реального мира, также должны меняться, чтобы адекватно отображать действительность. Поэтому для сопровождения и эксплуатации информационных систем требуется постоянное использование процедур проектирования БД.

Методология проектирования БД может рассматриваться как совокупность методов и средств, последовательное применение которых обеспечивает разработку проекта БД, удовлетворяющего заданным целям. Рассматриваемая методология позволяет пользователю лучше понять, как следует специфицировать требования к данным.

1.1. Основные понятия баз данных

Основным назначением ИС является оперативное обеспечение пользователя информацией о внешнем мире. Однако пользователя интересует не весь окружающий мир, а некоторый его фрагмент, который и находит свое воплощение в автоматизированной ИС.

Предметная область – отражение в ИС совокупности и объектов реального мира с их связями, относящимися к некоторой области знаний и имеющими практическую ценность для пользователя.

Понятие предметной области было введено в начале 80-х гг. прошлого века, когда учеными в области ИС была осознана необходимость использовать семантические модели для представления информации в компьютерных системах. Так же как требования к компьютерной системе формируются средствами естественного языка, так и информация в компьютерных системах представляется средствами особого языка с определенной семантикой. Такой подход впервые был представлен П. Ченом в 1976 г.

Исходя из понятия предметной области, перейдем к понятию БД. **База данных** объединяет в себе информационное представление объектов предметной области, связей между ними и операций над ними, образуя тем самым информационную и функциональную модели предметной области и описывая ее состояние с определенной точностью.

Информация о внешнем мире представляется в любой ИС, в том числе и в БД в форме данных. Это ограничивает возможности смысловой интерпретации информации и конкретизирует семантику ее представления в системе.

Попытаемся дать определение БД исходя из понятия «данные» и «информация».

Термин информация происходит от латинского слова informatio, что означает «сведения, разъяснения, изложение». **Информация** – любые

сведения, не известные ранее получателю информации, пополняющие его знания.

Одно и то же информационное сообщение может содержать разное количество информации для разных людей в зависимости от накопленных ими знаний, от уровня понимания этого сообщения и интереса к нему. Так, сообщение, составленное на японском языке, не несет никакой новой информации человеку, не знающему этого языка, но может быть высокоинформативным для человека, владеющего японским. Никакой новой информации не содержит и сообщение, изложенное на знакомом языке, если его содержание непонятно или уже известно.

Информация может существовать и передаваться в виде:

- текстов, рисунков, чертежей, фотографий;
- световых или звуковых сигналов;
- радиоволн;
- электрических и нервных импульсов;
- магнитных записей;
- жестов и мимики;
- запахов и вкусовых ощущений;
- хромосом и т. д.

Одним из важнейших видов информации является **экономическая информация**. Экономическая информация сопровождает процессы производства, распределения, обмена и потребления материальных благ и услуг. Экономическая информация – совокупность сведений, отражающих социально-экономические процессы и служащих для управления этими процессами и коллективами людей в производственной и непроизводственной сфере.

Информация есть характеристика не сообщения, а соотношения между сообщением и его потребителем. Без наличия потребителя, хотя бы потенциального, говорить об информации бессмысленно. Информация извлекается получателем из соответствующих данных.

Данные – информация о каком-либо событии, процессе или объекте, зафиксированная в определенной форме, пригодной для восприятия, передачи, преобразования, хранения или использования.

Так, написанные на листе 10 номеров телефонов в виде последовательности десяти чисел представляют собой данные, но не несут в себе информации. В то же время если напротив каждого номера указать фамилию человека и его адрес, то цифры обретут определенность и превратятся из данных в информацию.

Традиционно фиксация данных осуществляется при помощи конкретного средства общения (например, при помощи естественного языка или изображений) на конкретном носителе (например, бумаге или дискете). Обычно данные и их интерпретация (семантика) фиксируются совместно. Примером может служить утверждение «Оценка студента – 5». Здесь «5» – данное, а «Оценка студента» – его семантика.

В окружающем мире информация может быть представлена в виде данных разного типа: числовых, текстовых, графических, аудио, видео и т. п. Именно данные и являются предметом хранения в БД.

Первоначально (начало 60-х гг.) использовалась файловая система хранения данных. Для решения преимущественно инженерных задач, характеризующихся небольшим количеством данных и значительным объемом вычислений, данные хранились непосредственно в программе. Применялся последовательный способ организации данных с их высокой избыточностью, идентичностью логической и физической структур и полной зависимостью. С появлением экономико-управленческих задач, отличающихся большими объемами данных и малой долей вычислений, указанная организация данных оказалась неэффективной. Требовалось их упорядочение, которое, как выяснилось, возможно было проводить по двум критериям: использование (информационные массивы) и хранение (базы данных). Первоначально применяли информационные массивы, но вскоре выяснилось превосходство БД. Использование файлов для хранения только данных было предложено Мак Гри в 1959 г. Были разработаны методы доступа (в том числе произвольного) к таким файлам, при этом физическая и логическая структуры уже различались, а физическое расположение данных можно было менять без изменения логического представления.

В 1963 г. С. Бахманом была построена первая промышленная БД, при этом доступ к данным осуществлялся при помощи соответствующего программного обеспечения (ПО).

Таким образом, под **базой данных** понимают совокупность хранящихся вместе данных о конкретной предметной области при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений.

Целью создания БД как разновидности информационной технологии и формы хранения данных является построение системы данных, не зависящих от принятых алгоритмов и ПО, применяемых технических средств и физического расположения данных в ЭВМ, которые обеспечивают непротиворечивую и целостную информацию при нерегламентируемых запросах.

БД предполагает многоцелевое ее использование (несколько пользователей, множество форм документов и запросов одного пользователя). Основные функции базы данных: сбор, хранение, поиск, извлечение и представление данных.

1.2. Модели и виды баз данных

Модель данных – множество элементов (объектов, типов данных) и связей между ними, ограничений (например, целостности, синхронизации многопользовательского доступа, авторизации) операций над типами данных и отношениями.

Модель данных определяет логическая структуру данных, которая представляет присущие этим данным свойства, не зависимые от аппаратного и программного обеспечения и не связанные с функционированием компьютера.

Модель и структура БД зависит от выбранной модели представления данных. Для каждой модели данных создан специальный класс программных продуктов, позволяющие создавать и управлять БД. Этот класс ПО получил название **системы управления базами данных (СУБД).** Основная особенность СУБД – это наличие процедур для ввода и хранения не только самих данных, но и описаний их структуры.

Можно выделить следующие модели данных: иерархическая, сетевая, реляционная, объектно-ориентированная, многомерная.

Иерархическая модель

Иерархическая модель была исторически первой структурой БД, видимо, в связи с тем, что древовидные иерархические структуры широко используются в повседневной человеческой деятельности. Это всевозможные классификаторы, ускоряющие поиск требуемой информации, иерархические функциональные структуры управления.

Иерархическая модель состоит из объектов с указателями от родительских объектов к потомкам, соединяя вместе связанную информацию. Такие БД могут быть представлены как дерево, состоящее из объектов различных уровней. Верхний уровень занимает один объект, второй – объекты второго уровня и т. д.

Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня). При этом возможна ситуация, когда объект-предок не имеет потомков или имеет их несколько, тогда как у объекта-потомка обязательно только один предок.

Например, если иерархическая база данных содержала информацию о покупателях и их заказах, то будут существовать объект «покупатель» (родитель) и объект «заказ» (дочерний).

Достоинством такой модели является скорость поиска. Так запрос, направленный вниз по иерархии, прост (например: какие заказы принадлежат этому покупателю).

В то же время запрос, направленный вверх по иерархии, более сложен (например: какой покупатель поместил этот заказ). Кроме того, трудно представить неиерархические данные при использовании этой модели (подходит не для любой предметной области). Сложными становятся операции включения новых объектов и удаления устаревших объектов (в особенности обновление и удаление связей).

Наиболее известной иерархической СУБД до сих пор остается Information Management System (IMS) фирмы IBM.

Сетевая модель

Сетевые модели – это достаточно сложные структуры, состоящие из «наборов» – поименованных двухуровневых деревьев. «Наборы» соединяются при помощи «записей-связок», образуя цепочки и т. д.

К основным понятиям сетевой модели относятся: уровень, элемент (узел), связь. Узел – это совокупность атрибутов данных, описывающих некоторый объект. На схеме иерархического дерева узлы представляются вершинами графа. В сетевой структуре каждый элемент может быть связан с любым другим элементом. В сетевых БД имеются указатели в обоих направлениях, которые соединяют родственную информацию.

Несмотря на то, что эта модель решает некоторые проблемы, связанные с иерархической моделью, она имеет ряд недостатков:

– сложность самой модели данных влечет за собой и сложность выполнения запросов;

– поскольку логика процедуры выборки данных зависит от физической организации этих данных, эта модель не является полностью независимой от приложения.

При различных способах реализации сетевых моделей наибольшее распространение получила модель КОДАСИЛ (CODASYL). Эта модель считается наиболее развитой сетевой моделью данных, постоянно развивается, поддерживается и сопровождается, являясь как бы стандартом. Разрабатывались и соответствующие СУБД: DMS корпорации UNIVAC, IDMS (Cullinane), DBMS (DEC), IDS (Honeywell). В настоящее время широко известна db_Vista, работающая на персональных компьютерах в DOS и Windows.

Реляционная модель

Существенный скачок в развитии технологии баз данных произошел в связи с предложенной Э. Коддом в 1970 г. парадигмой реляционной модели данных. Теперь логические структуры могли быть получены из одних и тех же физических данных, т. е. доступ к одним и тем же физическим данным мог осуществляться различными приложениями по разным путям. Стало возможным обеспечение целостности и независимости данных.

Слово «реляционный» происходит от английского «relation» (отношение). В реляционных БД (РБД) все данные представлены в виде простых таблиц, разбитых на строки и столбцы, на пересечении которых расположены данные. Запросы к таким таблицам возвращают таблицы, которые сами могут становиться предметом дальнейших запросов. Каждая база данных может включать несколько таблиц.

К недостаткам РБД относят:

- для сложной предметной области РБД состоит из очень большого числа таблиц, связанных между собой весьма сложным образом. Разработчику реализация такой БД будет чрезвычайно сложной задачей;

- ПО, разработанное для РБД, обычно оказывается жестко завязанным на структуру реляционных таблиц. Если в дальнейшем будет принято решение изменить структуру таблиц, то все программное обеспечение, разработанное для прежней структуры таблиц, придется изменять.

Более подробно РБД будут рассмотрены в подразд. 1.4.

Объектно-ориентированная модель

Объектно-ориентированная база данных (ООБД) – база данных, в которой данные оформлены в виде моделей объектов, включающих прикладные программы, которые управляются внешними событиями.

Возникновение ООБД определяется прежде всего потребностями практики: необходимостью разработки сложных информационных прикладных систем с высокой производительностью.

Однако ООБД на современном уровне развития имеют пока много недостатков. Среди них: сложность структуры; отсутствие отработанного языка; отсутствие поддержки авторизации.

Имеется ряд коммерческих объектно-ориентированных СУБД. В их числе GemStone (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).

Многомерная модель

Многомерные модели рассматривают данные либо как факты с соответствующими численными параметрами, либо как текстовые измерения, которые характеризуют эти факты. В розничной торговле, к примеру, покупка – это факт, объем покупки и стоимость – параметры, а тип приобретенного продукта, время и место покупки – измерения. Запросы агрегируют значения параметров по всему диапазону измерения, и в итоге получают такие величины, как общий месячный объем продаж данного продукта.

Многомерные БД рассматривают данные как кубы, которые являются обобщением электронных таблиц на любое число измерений. Кроме того, кубы поддерживают иерархию измерений и формул без дублирования их определений. Набор соответствующих кубов составляет многомерную базу данных (или хранилище данных).

Многомерные модели данных имеют три важные области применения, связанные с проблематикой анализа данных:

- хранилища данных, интегрирующие для анализа информацию из нескольких источников на предприятии;
- системы оперативной аналитической обработки (online analytical processing – OLAP), позволяющие оперативно получить ответы на запросы, охватывающие большие объемы данных в поисках общих тенденций;
- приложения добычи данных (Data Mining), которые служат для выявления знаний за счет полуавтоматического поиска ранее не известных шаблонов и связей в базах данных.

Первым продуктом, выполняющим OLAP-запросы, был Express (компания IRI). Сейчас среди многомерных СУБД можно выделить D3 Server, UniVerse, Unidata и др.

Отдельные БД могут объединять все данные, необходимые для решения одной или нескольких прикладных задач, или данные, относящиеся к какой-либо предметной области (например, финансам, образованию, кулинарии и т.

п.). Первые обычно называют **прикладными БД**, а вторые – **предметными БД** (соотносящимся с предметами организации, а не с ее информационными приложениями). (Первые можно сравнить с базами материально-технического снабжения или отдыха, а вторые – с овощными и обувными базами.)

Предметные БД позволяют обеспечить поддержку любых текущих и будущих приложений, поскольку набор их элементов данных включает в себя наборы элементов данных прикладных БД. Такая гибкость и приспосабливаемость позволяет создавать на основе предметных БД достаточно стабильные информационные системы, т. е. системы, в которых большинство изменений можно осуществить без вынужденного переписывания старых приложений.

Основывая же проектирование БД на текущих и предвидимых приложениях, можно существенно ускорить создание высокоэффективной информационной системы, т. е. системы, структура которой учитывает наиболее часто встречающиеся пути доступа к данным. Поэтому прикладное проектирование до сих пор привлекает некоторых разработчиков. Однако по мере роста числа приложений таких информационных систем быстро увеличивается число прикладных БД, резко возрастает уровень дублирования данных и повышается стоимость их ведения.

В настоящее время существует большое число программных продуктов, относящихся к этому классу. Это и открытые СУБД, используемые для небольших проектов и малых компаний (PostgreSQL, MySQL), и целые платформы (Oracle, MSSQL, DB2, Sybase) для крупных проектов.

1.3. Системы управления базами данных

Постоянное изменение основных свойств вычислительной техники и развитие СУБД обусловливали возникновение различных моделей взаимодействия пользователей с БД. Дадим краткую характеристику этим моделям в хронологическом порядке.

Модель с централизованной архитектурой

При использовании этой технологии БД, СУБД и прикладную программу (приложение) располагают на одном компьютере. Для такого способа организации не требуется поддержки сети, и все сводится к автономной работе.

Работа с этой моделью построена следующим образом. На компьютере хранится БД, установлены СУБД и приложение для работы с БД. Пользователь запускает приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД. Все обращения идут через СУБД, которая хранит внутри себя все сведения о физической структуре БД. СУБД инициирует обращения к данным, обеспечивая выполнение запросов пользователя. Результат СУБД возвращает в приложение, а приложение отображает результат выполнения запросов. Таким образом, в этой модели реализуется однопользовательский режим работы.

В данной модели реализуем и многопользовательский режим. С этой целью к мэйнфрейму подключалось несколько терминалов, но тогда приходилось обслуживать в рамках ресурсов одного компьютера весь комплекс

возникающих задач – от обработки и хранения данных до отображения информации и приема запросов пользователей. Таким образом, основным недостатком этой модели является резкое снижение производительности при увеличении числа пользователей.

Подобная архитектура использовалась в первых версиях СУБД DB2, Oracle, Ingres.

Модель с автономными персональными ЭВМ

Каждый пользователь имеет свою автономную персональную ЭВМ. База данных и СУБД копируются на компьютере каждого пользователя, и он работает с базой данных на своей ЭВМ.

Для данной модели характерна полная децентрализация данных. Основным недостатком модели является невозможность оперативного обновления данных на всех компьютерах при изменении их одним из пользователей.

Архитектура «файл–сервер»

Эта архитектура предполагает назначение одного из компьютеров сети в качестве выделенного сервера, на котором будут храниться файлы базы данных. В соответствии с запросами пользователей файлы с «файл–сервера» передаются на рабочие станции пользователей, где и осуществляется основная часть обработки данных. Центральный сервер выполняет в основном только роль хранилища файлов, не участвуя в обработке самих данных.

В этой модели БД в виде набора файлов находится на жестком диске файлового сервера. Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлены СУБД и приложение для работы с БД. На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, они через СУБД инициируют обращение к БД. СУБД инициирует обращения к данным, находящимся на файловом сервере, в результате которых часть файлов БД копируется на клиентский компьютер и обрабатывается, что обеспечивает выполнение запросов пользователя.

В случае изменения данные отправляются назад на файловый сервер для обновления БД. Результат СУБД возвращает в приложение. Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Основным недостатком модели является то, что при одновременном обращении множества пользователей к одним и тем же данным производительность работы системы резко падает из-за необходимости ждать, пока пользователь, работающий с данными, завершит свою работу. Кроме того, вся тяжесть вычислительной нагрузки при доступе к БД ложится на приложение клиента, так как при выдаче запроса на выборку информации из таблицы вся таблица БД копируется на клиентскую машину и выборка осуществляется на клиенте. Таким образом, не оптимально расходуются ресурсы клиентского компьютера и сети. В результате возрастает сетевой трафик и увеличиваются требования к аппаратным мощностям пользовательского компьютера.

Примеры: dBase, Microsoft Access, FoxPro, Paradox.

Архитектура «клиент–сервер»

Использование архитектуры «клиент–сервер» предполагает наличие некоторого количества компьютеров, объединенных в сеть, один из которых выполняет особые управляющие функции (является сервером сети). Архитектура «клиент–сервер» разделяет функции приложения пользователя (клиента) и сервера. Приложение-клиент формирует запрос к серверу, на котором расположена БД. Удаленный сервер принимает запрос и переадресует его серверу БД.

Работа построена следующим образом. БД в виде набора файлов и СУБД находятся на жестком диске сервера сети. Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлено клиентское приложение для работы с БД. На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый интерфейс, он инициирует обращение к СУБД (по сети от клиента к серверу передается лишь текст запроса). СУБД хранит внутри себя все сведения о физической структуре БД, расположенной на сервере. СУБД инициирует обращения к данным, находящимся на сервере, где осуществляется вся обработка данных, и лишь результат выполнения запроса копируется на клиентский компьютер. Таким образом, СУБД возвращает результат в приложение. Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Клиент-серверные СУБД, в отличие от файл-серверных, обеспечивают разграничение доступа между пользователями и мало загружают сеть и клиентские машины.

Недостаток клиент-серверных СУБД – в самом факте существования сервера (что плохо для локальных программ) и больших вычислительных ресурсах, потребляемых сервером. Кроме того, большое количество клиентских компьютеров, расположенных в разных местах, вызывает определенные трудности со своевременным обновлением клиентских приложений на всех компьютерах-клиентах.

Примеры: Firebird, Interbase, MS SQL Server, Oracle, PostgreSQL, Gupta, Informix, Sybase, DB2.

Трехзвенная (многозвенная) архитектура

В трехзвенной архитектуре вся бизнес-логика, ранее входившая в клиентские приложения, выделяется в отдельное звено, называемое сервером приложений. При этом клиентским приложениям остается лишь пользовательский интерфейс. Так, в качестве клиентского приложения может выступать Web-браузер.

В результате работа построена следующим образом. БД и СУБД располагаются на сервере сети. Существует специально выделенный сервер приложений, на котором располагается программное обеспечение (ПО) делового анализа (бизнес-логика). На каждом из множества клиентских компьютеров установлен так называемый «тонкий клиент» – клиентское приложение, реализующее интерфейс пользователя. Используя

предоставляемый приложением интерфейс, пользователь инициирует обращение к ПО делового анализа, расположенному на сервере приложений.

Сервер приложений анализирует требования пользователя и формирует запросы к БД. СУБД хранит внутри себя все сведения о физической структуре БД, расположенной на сервере. СУБД инициирует обращения к данным, находящимся на сервере, в результате которых результат выполнения запроса копируется на сервер приложений. Сервер приложений возвращает результат в клиентское приложение (пользователю). Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Таким образом, теперь при изменении бизнес-логики более нет необходимости изменять клиентские приложения и обновлять их у всех пользователей. Кроме того, максимально снижаются требования к аппаратуре пользователей.

Встраиваемые СУБД

Встраиваемая СУБД – библиотека, которая позволяет унифицированным образом хранить большие объемы данных на локальной машине. Доступ к данным может происходить через SQL либо через особые функции СУБД. Встраиваемые СУБД быстрее обычных клиент-серверных и не требуют установки сервера, поэтому востребованы в локальном ПО, которое имеет дело с большими объемами данных (например, геоинформационные системы).

Примеры: OpenEdge, SQLite, BerkeleyDB, Sav Zigzag, Progress Enterprise Server, Centura SQLBase, IB Database, Sybase SQL Anywhere, Empress IMC.

1.4. Реляционные базы данных

В конце 60-х гг. появились работы, в которых обсуждались возможности применения различных табличных моделей данных, т. е. возможности использования привычных и естественных способов представления данных. Наиболее значительной из них была статья сотрудника фирмы IBM Э. Ф. Кодда (Codd E.F., A Relational Model of Data for Large Shared Data Banks. CACM 13: 6, June 1970), где впервые был применен термин «реляционная модель данных».

Будучи математиком по образованию, Э. Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение – relation (англ.)

Основным понятием РБД является сущность. **Сущность** (entity) – любой объект предметной области, информацию о котором необходимо хранить в БД. Сущность предметной области является результатом абстрагирования реального объекта предметной области путем выделения и фиксации набора его свойств (атрибутов). **Атрибуты сущности** – поименованные свойства, характеризующие сущность. Например, сущность СТУДЕНТ имеет атрибуты ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО, № СТУДЕНЧЕСКОГО БИЛЕТА и т. п.

Сущности вступают в связи друг с другом через свои атрибуты. Каждая группа атрибутов, описывающих одно реальное проявление сущности, представляет собой экземпляр (instance) сущности. Иными словами,

экземпляры сущности – это реализации сущности, отличающиеся друг от друга и допускающие однозначную идентификацию.

Связь – ассоциирование двух или более сущностей. Если бы назначением БД было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации БД – это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи. А так как в реальных БД нередко содержатся сотни или даже тысячи сущностей, то теоретически между ними может быть установлено более миллиона связей. Наличие такого множества связей и определяет сложность логических моделей.

В дальнейшем будем рассматривать РБД как совокупность сущностей, отношений и связей между ними, содержащих всю информацию из предметной области. Однако пользователи могут воспринимать такую базу данных как совокупность таблиц.

Поскольку сущности в реляционной БД хранятся в форме таблиц, необходимо указать на соответствие терминов и сферы их использования.

Как видно из табл. 1.1, атрибуты сущности хранятся в столбцах таблицы и называются **поля**, а данные (экземпляры сущности) хранятся в строках и называются **записи.**

Таблица 1.1

Предметная область	Реляционная модель	Табличная структура
Сущность	Таблица	Таблица
Атрибут	Поле	Столбец
Экземпляр	Запись	Строка

Таблицы реляционной БД обладают следующими свойствами:

1. Каждая таблица состоит из однотипных строк и имеет уникальное имя.

2. Все записи должны иметь одинаковую структуру, т. е. одно и то же количество полей с соответственно совпадающими именами.

3. Все поля должны быть атомарны (неделимы), т. е. в каждой позиции таблицы на пересечении строки и столбца всегда имеется в точности или одно значение или нет ничего.

4. Все записи должны быть уникальны.

5. Поля имеют уникальные имена, и в каждом из них размещаются однородные (однотипные) значения данных.

Для манипулирования таблицами Э. Ф. Кодд создал инструмент – **реляционную алгебру**. Каждая операция этой алгебры использует одну или несколько таблиц в качестве ее операндов и продуцирует в результате новую таблицу, т. е. позволяет «разреза́ть» или «склеивать» таблицы.

Один из основных вопросов, который может возникнуть на этом этапе: почему нельзя обойтись одной большой таблицей, в которой будет храниться вся информация обо всех сущностях предметной области?

Рассмотрим пример. Пусть имеется таблица ПРЕПОДАВАТЕЛИ (табл. 1.2). На первый взгляд, данная таблица вполне хороша, так как хранит в себе все требуемые атрибуты.

Таблица 1.2. Преподаватели

Фамилия	*Должность*	*Кафедра*	*Телефон*	*Предмет*	*Часы*
Иванов	Доцент	ИТЭ	351	Информатика	100
Петров	Профессор	ИТЭ	351	Базы данных	200
Петров	Профессор	ИТЭ	351	Информатика	100

Однако при использовании такой универсальной таблицы возникает ряд проблем (аномалий):

1. **Аномалия обновления.** Проблема, связанная с избыточностью данных. Данные практически всех полей многократно повторяются. Повторяются и некоторые наборы данных (Фамилия–Должность–Кафедра–Телефон, Предмет–Часы). Это приводит к тому, что одни и те же данные необходимо вводить несколько раз, что многократно увеличивает время ввода и может привести к опечаткам. Кроме того, при необходимости изменения номера телефона кафедры его придется изменять в таблице многократно.

2. **Аномалия вставки (включения).** Невозможность ввести данные в таблицу ввиду отсутствия других данных. В таблицу невозможно добавить сведения о преподавателе, если он не ведет никакой предмет.

3. **Аномалия удаления.** Непреднамеренная потеря данных в связи с удалением других данных. Так, при удалении сведений о преподавателе (например, при его увольнении) пропадают все данные о предмете, который он вел.

Для того чтобы устранить аномалии, необходимо провести процедуру нормализации. **Нормализация** – пошаговый процесс разложения (декомпозиции) исходных таблиц на более простые таблицы, связанные между собой.

Принципиально важным достоинством реляционных БД является, то, что связи между таблицами не зависят от физической структуры данных. Таким образом, они являются не физическими, привязанными к местоположению данных (как, например, при использовании табличных процессоров, где значение ячейки A1 одного листа связывается со значением ячейки B2 другого листа), а **логическими**. В связи с этим при выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию.

Между двумя сущностям, например A и B, возможны четыре вида логических связей. В БД они называются **структурными** связями.

«Один-к-одному» (1:1)

Каждой записи таблицы A соответствует только одна запись таблицы B, и наоборот.

Если между двумя таблицами связь 1:1, то эти таблицы можно объединить в одну. Обычно связь используется для разделения доступа к данным для различных пользователей.

«Один-ко-многим» (1:М)

Каждой записи таблицы А может соответствовать несколько записей таблицы В, а одной записи таблицы В соответствует только одна запись таблицы А.

«Многие-к-одному» (М:1)

Каждой записи таблицы А соответствует только одна запись таблицы В, а одной записи таблицы В может соответствовать несколько записей таблицы А.

«Многие-ко-многим» (М:М)

Каждой записи таблицы А может соответствовать несколько записей таблицы В, а каждой записи таблицы В может соответствовать несколько записей таблицы А.

Рассмотрим в качестве примера сущности СТУДЕНТЫ и АДРЕСА:

1) связь между данными сущностями будет 1:1, в случае если один студент имеет только один адрес, то по одному адресу может проживать только один студент;

2) в случае если в БД фиксируется адрес прописки и адрес проживания, то получим связь 1:М, так как один студент может иметь несколько адресов, а по одному адресу проживает только один студент;

3) в случае если в БД фиксируется только адрес проживания, но возможна ситуация, когда студенты проживают в общежитии (т. е. по одному адресу), получим связь М:1;

4) если возможна ситуация и п. 2 и п. 3, то получим связь М:М.

Как видно из данного примера в одной и той же предметной области могут быть разные логические связи между одними и теми же сущностями. Поэтому только Заказчик (пользователь) БД, который разбирается в предметной области, может их правильно определить на этапе проектирования.

Для представления типов связи обычно используют **ER-диаграммы** (от англ. Entity-Relationship, т. е. сущность-связь). В СУБД MS Access ER-диаграмма представлена в виде **Схемы данных.**

Для реализации связей используются специальные **ключевые поля.** Различают первичные и внешние ключи.

Первичный ключ – одно или несколько полей таблицы, однозначно идентифицирующих каждую запись.

Первичный ключ обладает следующими свойствами:

- значения первичного ключа не могут повторяться;
- первичный ключ не может принимать неопределенное значение (быть пустым).

В случае если в таблице нет ни одного поля, подходящего под эти требования можно создать **составной первичный ключ**, состоящий из нескольких полей, которые в своей совокупности не повторяются. При этом необходимо соблюдать требование минимальности: ни один из атрибутов не может быть исключен из набора без нарушения уникальности.

В качестве первичных ключей нежелательно использовать поля, содержащие длинный текст (например, название вуза). В этом случае проще добавить в таблицу новое поле (например, код вуза).

Также не рекомендуется создавать составные ключи, состоящие из более чем трех полей. Такая конструкция будет неудобна в использовании.

Внешний ключ – одно или несколько полей, связанных с первичным ключом из другой таблицы. При этом данные во внешнем ключе могут повторяться.

При создании связей первичные и внешние ключи используются следующим образом:

1) *«один-к-одному»* – в таблицах А и В создаются одинаковые первичные ключи;

2) *«один-ко-многим»* и *«многие-к-одному»* – на стороне «один» создается первичный ключ, а на стороне «много» – соответствующий ему внешний ключ;

3) *«многие-ко-многим»* – связь М:М невозможно реализовать напрямую. Для установления связи используют третью таблицу, которую будем называть отношением. **Отношение** – это процесс или событие, в котором участвуют несколько сущностей.

Приведем примеры, используя сущности СТУДЕНТЫ и АДРЕСА.

«Один-к-одному» (рис. 1.1)

В таблице Студенты создадим первичный ключ № студ. билета. В таблице Адреса создадим такой же первичный ключ и соединим их между собой.

Рис. 1.1

«Один-ко-многим» (рис. 1.2)

В таблице Студенты создадим первичный ключ № студ. билета. В таблице Адреса создадим такое же поле, но которое будет являться внешним ключом (не уникальным), и соединим их между собой.

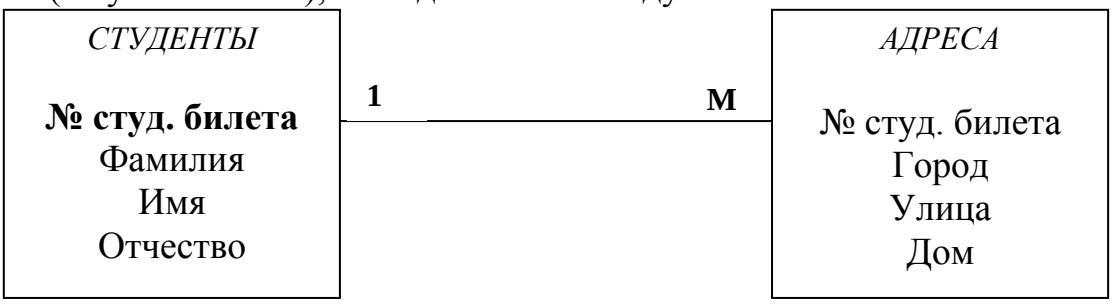

Рис. 1.2

«Многие-к-одному» (рис. 1.3)

В таблице Адреса создадим новое поле первичного ключа Код Адреса. В таблице Студенты создадим такое же поле, но оно будет являться внешним ключом, и соединим их между собой.

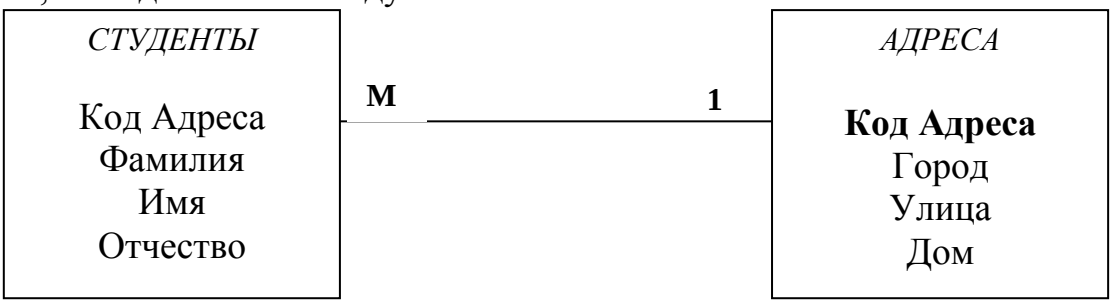

Рис. 1.3

«Многие-ко-многим» (рис. 1.4)

В таблице Студенты создадим первичный ключ № студ. билета. В таблице Адреса создадим первичный ключ Код Адреса. Создадим третью таблицу-отношение и назовем ее ПРОЖИВАНИЕ. Добавим в эту таблицу внешние ключи № студ. билета и Код Адреса и свяжем соответствующие пары первичных и внешних ключей.

При этом получим еще пару связей 1:М между таблицами Студенты–Проживание и Адреса–Проживание.

Одной из основных ошибок при определении связей является проверка типа связи только в одну сторону. Например, делается вывод о том, что между сущностями СТУДЕНТЫ и АДРЕСА связь «один-ко-многим», так как один студент может иметь несколько адресов. При этом забывается необходимость проверки от Адресов к Студентам (по одному адресу может проживать несколько студентов), что приводит к неправильной модели предметной области и, как следствие, к неработоспособной БД.

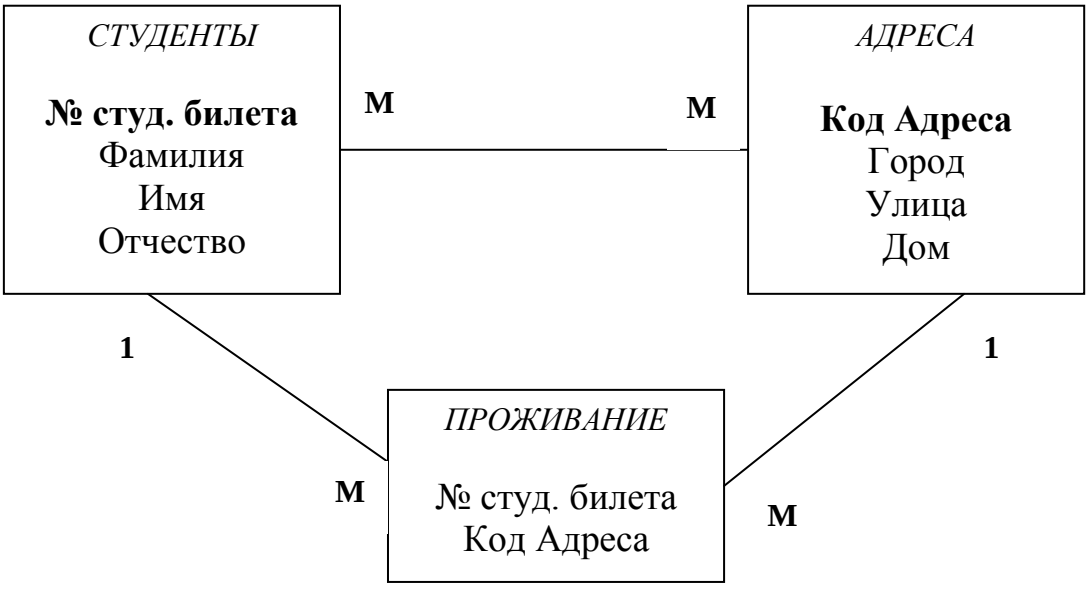

Рис. 1.4

Вернемся к нашему примеру (табл. 1.2) и проведем нормализацию таблицы (результат нормализации представлен на рис. 1.5).

Во-первых, выделим независимые сущности: ПРЕПОДАВАТЕЛИ, КАФЕДРЫ и ПРЕДМЕТЫ. Затем определим связи между ними.

На одной кафедре могут работать несколько преподавателей, но один преподаватель может работать только на одной кафедре. Следовательно, связь между Кафедрами и Преподавателями «один-ко-многим».

Один предмет могут вести несколько преподавателей, а один преподаватель может вести несколько предметов. Следовательно, связь между Преподавателями и Предметами – «многие-ко-многим».

Реализуем выявленные связи. Для связи Кафедры–Преподаватели определим в качестве первичного ключа таблицы Кафедры (сторона «один») поле **№ кафедры**. Несмотря на то что Название кафедры тоже можно было бы использовать в качестве первичного ключа, оно содержит длинный текст, и поэтому неудобно в использовании. В таблице Преподаватели создадим такое же поле, но выступающее уже в качестве внешнего ключа и соединим их между собой.

Как было уже сказано, связь «многие-ко-многим» реализуется через таблицу-отношение. В качестве такой таблицы используем **Расписание**. В таблице Предметы создадим новое поле первичного ключа **Код Предмета**. В таблице Преподаватели первичным ключом будет новое поле **Личный Код.** Фамилия преподавателя не может быть первичным ключом, так как может содержать повторяющиеся данные. Добавим соответствующие внешние ключи в таблицу Расписание и свяжем соответствующие пары первичных и внешних ключей.

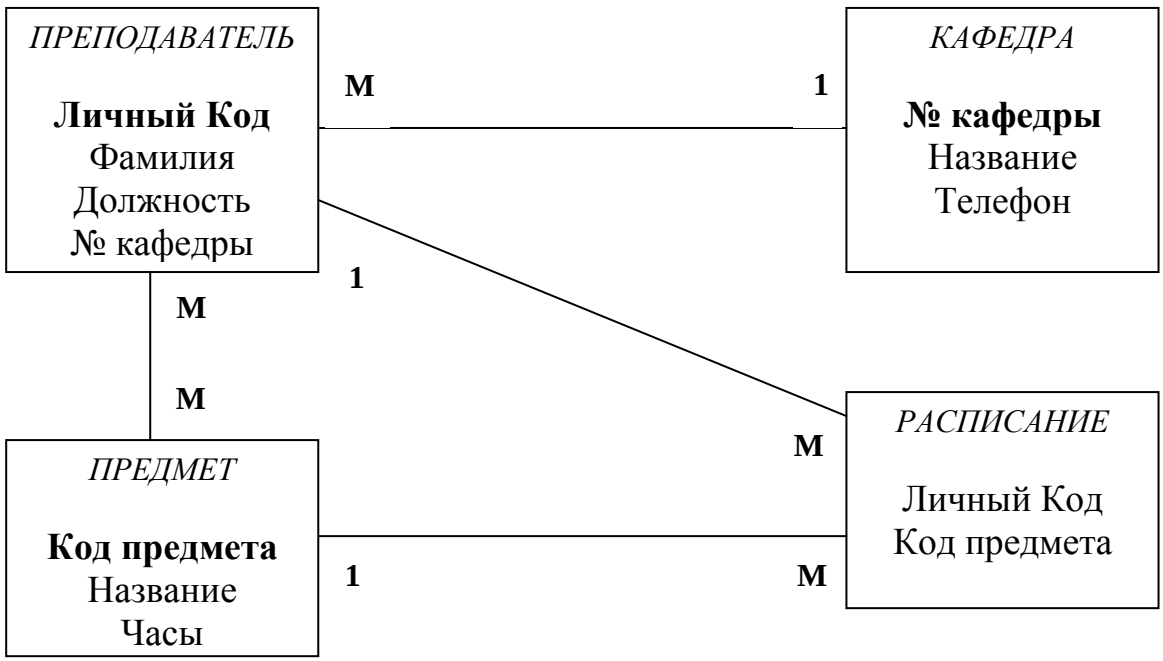

Рис. 1.5

Теперь проверим созданную ER-модель на аномалии. Аномалия обновления отсутствует, так как избыточность была ликвидирована, и все данные хранятся только один раз. Аномалия вставки ликвидирована, так как мы можно добавлять данные о новых предметах и преподавателях независимо друг от друга. Аномалия удаления также ликвидирована, так как удаление,

например, сведений о предмете не приводит к удалению сведений о преподавателе.

Одна из основных проблем в любой модели БД – это проблема целостности данных. **Целостность данных** – устойчивость хранимых данных к неверным изменениям или разрушению, связанным с ошибочными действиями пользователей.

Целостность данных предполагает:

1) надежность данных – отсутствие неточно введенных данных или двух одинаковых записей об одном и том же факте;

2) ссылочную целостность – защиту данных при удалении или обновлении связанных данных разных таблиц.

Надежность данных достигается соблюдением двух условий: достоверности и непротиворечивости.

Достоверность данных – правильность данных в любой момент времени. БД не может контролировать правильность каждого отдельного вводимого значения (хотя каждое значение можно проверить на правдоподобность). Например, нельзя обнаружить, что вводимое значение 5 (представляющее номер месяца) в действительности должно быть равно 3. С другой стороны, значение 20 явно будет ошибочным.

Непротиворечивость данных – отсутствие противоречия между введенными данными и предметной областью, а также между вводимыми данными в разные поля одной записи.

Рассмотрим значение «100», вводимое в поле Год рождения. Данное значение достоверно, так как это число целое, и такой год рождения вполне мог существовать. Сложнее дело обстоит с проверкой на непротиворечивость. Если предметной областью БД является экономический факультет СПбГУТ, то вводимое значение противоречит предметной области, если же предметная область – античные философы, то значение непротиворечиво. Однако на этом проверка не закончена. Даже если БД посвящена античным философами, но в обновляемой записи в поле Год смерти стоит число «90», то вводимые данные также противоречивы (число года рождения не может быть больше числа года смерти). Условие на непротиворечивость закладываются непосредственно заказчиком БД на этапе проектирования.

Ссылочная целостность определяет систему правил для поддержания связей между записями в связанных таблицах.

При связи «один-ко-многим» актуальными становятся вопросы:

- что делать при удалении записи в таблице на стороне «один»?
- что делать при изменении данных в поле первичного ключа?

Для обоих случаев есть два варианта: ограничение операции или ее каскадирование.

В случае **ограничения** пользователь не сможет изменить данные в первичном ключе, если есть связанная запись в другой таблице, а также не сможет удалить запись в таблице на стороне «один», если есть связанная запись на стороне «много». Это делается для того, чтобы, во-первых, не потерять связь

между записями, а во-вторых, не получить «висячую» запись (при удалении сведений о преподавателе должны исчезнуть ссылки на него в расписании).

При **каскадировании** происходит автоматическое обеспечение ссылочной целостности. Изменение значения в первичном ключе приводит к автоматическому обновлению соответствующих значений во всех внешних ключах. Удаление же записи в главной таблице (сторона «один») приводит к автоматическому удалению связанных записей в подчиненных таблицах (сторона «много»).

1.5. Жизненный цикл базы данных

Организация структуры БД формируется исходя из следующих соображений:

– адекватность описываемому объекту/системе должна быть на уровне концептуальной и логической моделей;

– удобство использования для ведения учета и анализа данных должно быть на уровне физической модели.

Концептуальная модель – модель предметной области, состоящей из перечня взаимосвязанных понятий, используемых для описания этой области (вместе со свойствами и характеристиками, классификацией этих понятий по типам, ситуациям, признакам в данной области), и законов протекания процессов в ней. Концептуальная модель разрабатывается без какой-либо ориентации на конкретную СУБД, ее цель – получение семантических (смысловых) моделей, отражающих информационное содержание предметной области.

Логическая модель описывает понятия предметной области и их взаимосвязи и является прототипом будущей БД. Она представляет собой запись концептуальной модели на языке СУБД. Ее цель – описать объекты предметной области абстрактными понятиями модели данных так, чтобы это описание не противоречило семантике предметной области и было по возможности наилучшим.

Основным понятием логической модели является сущность.

Физическая модель является компьютерно-ориентированной моделью и описывает физическое расположение хранимых данных, процедур и методов доступа к ним.

Таким образом, при проектировании БД решаются две основных проблемы:

- каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было по возможности лучшим. Часто эту проблему называют проблемой логического проектирования;
- как обеспечить эффективность выполнения запросов к базе данных, т. е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создание каких дополнительных структур (например, индексов) потребовать и т. д. Эту проблему называют проблемой физического проектирования баз данных.

В данном пособии, говоря о проектировании, будем иметь в виду только концептуальное и логическое проектирование. Вопросы физического проектировании будут частично рассмотрены в разд. 2, посвященном созданию БД.

Жизненный цикл базы данных охватывает следующие основные этапы:

1) *проектирование* – происходит построение концептуальной и логической модели БД. Проектирование осуществляется заказчиком (конечным пользователем) БД. Этап проектирования полностью осуществляется «на бумаге» и никак не связан с использованием СУБД;

2) *создание* (реализация) БД – осуществляется разработчиком на компьютере посредством конкретной СУБД. Основой для создания является техническое задание, составленное заказчиком на предыдущем этапе;

3) *наполнение* – осуществляются конвертирование и загрузка данных, а также ввод данных. Под конвертированием данных понимают перенос любых существующих данных в новую БД. Этот шаг выполняется только в том случае, если новая БД заменяет старую. В противном случае данный этап предполагает непосредственный ввод данных;

4) *тестирование* – выполняется в целях поиска ошибок. Прежде чем использовать новую БД на практике, ее следует тщательно протестировать. Этого можно добиться путем разработки продуманной стратегии тестирования с использованием реальных данных, которая должна быть построена таким образом, чтобы весь процесс тестирования выполнялся строго последовательно и методически правильно;

5) *эксплуатация и сопровождение* – основной этап, включающий в себя использование БД и поддержка ее нормального функционирования по окончании развертывания;

6) *модернизация* – в процессе эксплуатации могут появиться новые функции, которые не были автоматизированы в БД, в этом случае на этапе модернизации может возникнуть необходимость создания нового шаблона документа, нового запроса и т. д. Важно подчеркнуть, что речь идет только о небольших добавлениях в первоначальный проект. Если же кардинально изменилась предметная область, можно говорить о необходимости проектирования новой БД.

2. ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

2.1. Этапы проектирования базы данных

Этап проектирования является первым и одним из важнейших этапов жизненного цикла БД. От корректного проектирования зависит то, насколько эффективно БД будет реализовывать свою цель – автоматизировать бизнес-процессы предприятия. Неоптимальные решения и прямые ошибки, заложенные на этапе проектирования, впоследствии очень трудно устраняются, поэтому этот этап является основополагающим.

Методически правильно начинать работу с карандашом и листом бумаги в руках, не используя компьютер. На данном этапе он просто не нужен. Проектирование БД всегда осуществляется заказчиком и заканчивается написанием технического задания (ТЗ) для разработчика. Однако для этого заказчик должен владеть соответствующей терминологией и знать, хотя бы в общих чертах, технические возможности основных СУБД. К сожалению, на практике это встречается не всегда. В связи с этим обычно используют следующие подходы:

- демонстрируют заказчику работу аналогичной БД, после чего согласовывают спецификацию отличий;
- если аналога нет, выясняют круг задач и потребностей заказчика, после чего разработчик помогает ему подготовить техническое задание.

Существует два основных подхода к проектированию БД: «**нисходящий**» и «**восходящий**».

При восходящем подходе работа начинается с самого нижнего уровня – уровня определения атрибутов сущностей, которые на основе анализа существующих между ними связей группируются в отношения, представляющие типы сущностей и связи между ними.

Восходящий подход лучше всего подходит для проектирования простых БД с относительно небольшим количеством атрибутов. Однако использование этого подхода существенно усложняется при проектировании БД с большим количеством атрибутов, установить среди которых все существующие функциональные зависимости довольно затруднительно.

Более подходящей стратегией проектирования сложных БД является использование нисходящего подхода. Начинается этот подход с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним атрибутов. Нисходящий подход демонстрируется в концепции модели «сущность–связь». В этом случае работа начинается с идентификации сущностей и связей между ними, интересующих данную организацию в наибольшей степени.

Помимо этих подходов для проектирования могут применяться другие подходы, например типа **«изнутри наружу»** или **«смешанная стратегия проектирования»**. Подход типа «изнутри наружу» похож на восходящий подход, но отличается от него начальной идентификацией набора основных сущностей с последующим расширением круга рассматриваемых сущностей,

связей и атрибутов, которые взаимодействуют с первоначально определенными сущностями. В смешанной стратегии сначала восходящий и нисходящий подходы используются для разных частей модели, после чего все подготовленные фрагменты собираются в единое целое.

Рассмотрим более подробно нисходящий подход. Этап проектирования состоит из нескольких шагов.

1. Описание предметной области.

Заказчик, как лицо, обладающее всей полнотой информации о протекающих бизнес-процессах и информационных потоках в компании, должен описать ту сферу деятельности, которая подлежит автоматизации. Здесь должны быть выявлены основные сущности, участвующие в описываемых бизнес-процессах, и степень их детализации, а также варианты использования этих сущностей (сами бизнес-процессы).

При этом очень важно не ограничиваться взаимодействием с головным подразделением заказчика, а провести обсуждение со всеми службами и подразделениями, которые могут оказаться поставщиками данных в базу или их потребителями.

Необходимая для проектирования информация может быть получена посредством:

- опроса и анкетирования отдельных сотрудников предприятия, особенно ведущих специалистов в наиболее важных областях ее деятельности;
- наблюдения за деятельностью предприятия;
- изучения документов, которые используются для сбора и представления информации;
- использования опыта проектирования других систем.

Рассмотрим в качестве примера предметную область «Приемная комиссия вуза». Основным заказчиком подобной БД может быть ректор вуза. В результате его интервьюирования могут быть получены данные о том, что основными сущностями являются: Абитуриенты, Специальности и Экзамены.

Основными вариантами использования будут являться: сбор анкетных данных об абитуриентах, фиксация оценок, полученных на экзаменах, расчет проходных баллов, зачисление студентов.

2. Построение ER-модели.

На этом шаге заказчику, скорее всего, понадобится помощь разработчика, который мог бы помочь сформулировать и нормализовать связи, отражающие схему информационного взаимодействия между сущностями в понятных ему терминах. Завершением данного шага является построение ER-модели.

В данном примере можно задать следующие вопросы:

Сколько абитуриентов может поступить на обучение по одной специальности?

На обучение по какому количеству специальностей может поступать один абитуриент?

Сколько экзаменов сдается при поступлении на обучение по разным специальностям одним абитуриентом?

Таким образом, получим следующие связи: Абитуриенты–Специальности – M:M, Абитуриенты–Экзамены – 1:M, Специальности–Экзамены – 1:M

3. Определение круга пользователей базы данных.

Необходимо учесть, что при проектировании важны даже не столько конкретные пользователи, сколько роли, которые они играют. Если есть три методиста с одинаковыми функциями и им всем нужен доступ к БД, то получаем лишь одну роль.

В этом примере, БД может быть полезна не только приемной комиссии, но и всем деканатам, службе, отвечающей за проживание студентов в общежитии, студенческому отделу кадров, военно-учетному столу, бухгалтерии.

4. Определение информационных потребностей пользователей.

Составив список пользователей БД, необходимо подробно описать все их (кого?) функции, которые могут быть автоматизированы при помощи БД. Для этого необходимо провести подробное интервьюирование всех пользователей и выписать из полного перечня их функций (например, из должностной инструкции) те, которые можно автоматизировать.

Например, формирование приказа на зачисление студентов может существенно повысить эффективность всех вышеперечисленных подразделений, так как избавит от необходимости многократного создания такого списка вручную.

5. Составление списка требуемых данных.

Для реализации каждой из функций нужно большое количество данных. Основная задача при этом – не включать в БД те данные, которые никому не нужны, а с другой стороны, не забыть то, что может понадобиться пользователям.

С этой целью для каждой из функций желательно получить списки:

- исходных данных, с которыми работают пользователи;
- выходных данных, которые необходимы пользователям для внутренней работы;
- выходных данных, которые не являются необходимыми для заказчика, но которые он должен предоставлять в другие организации (например, в городскую приемную комиссию).

Для составления такого перечня необходимо внимательно рассмотреть все используемые входные и выходные документы. Например, если для сдачи итоговой формы в министерство необходимо знать средний балл аттестата каждого абитуриента, то такие данные должны быть предусмотрены в БД.

Часть данных (например, фамилия абитуриента) будет повторяться для разных функций, поэтому необходимо составить список неповторяющихся данных (атрибутов сущностей), снабдив их кратким описанием. Отдельно стоит отметить те данные, которые могут быть получены на основании других данных путем вычислений.

В данном примере в результате проектирования был получен следующий список данных:

Фамилия, Имя, Отчество, Дата рождения, Пол абитуриента.

Название факультета, Название специальности, Шифр специальности.
Оценка, полученная на экзамене, Дата экзамена.

2.2. Этапы создания базы данных

2.2.1. Выбор СУБД

После получения ТЗ от пользователя, в котором представлена ER-модель и списки требуемых данных, разработчику необходимо перейти к реализации проекта на компьютере. При этом первой задачей является выбор используемой реляционной СУБД.

Рассмотрим процесс создания БД на основе реляционной СУБД MS Access. Microsoft Access в настоящее время является одной из самых популярных среди настольных СУБД. Однако, как и все файл-серверные СУБД он не лишен ряда недостатков. Не вызывают сомнения упреки в плохой защищенности данных, медленной работе, проблемах с дублированием данных при резервном копировании, трудностях администрирования, катастрофическом снижении скорости обработки при возрастании объемов данных и т. п.

Однако используемые для решения проблемы средства должны соответствовать сложности самой проблемы. Так, вряд ли имеет смысл тратить на разработку и внедрение информационной системы средства, существенно большие, чем весь годовой оборот предприятия, а для многих предприятий сферы малого и среднего бизнеса дело обстоит именно так.

Итак, где же используется на сегодняшний день MS Access? Прежде всего в государственных (муниципальных) учреждениях, сфере образования, сфере обслуживания, малом и среднем бизнесе. Специфика возникающих задач заключается в том, что объемы данных не являются катастрофически большими, частота обновлений не бывает слишком высокой, организация территориально обычно расположена в одном небольшом здании, количество пользователей колеблется от одного до 10–15 человек. В подобных условиях использование настольных СУБД для управления информационными системами является вполне оправданным, и они с успехом применяются.

Среди особенностей MS Access следует отметить:

- высокую степень универсальности и продуманности интерфейса, который рассчитан на работу с пользователями самой различной квалификации. В частности, реализована система управления объектами БД, позволяющая гибко и оперативно переходить из режима конструирования в режим их непосредственной эксплуатации;
- глубоко развитые возможности интеграции с другими программными продуктами, входящими в состав Microsoft Office (публикация в Word, анализ в Excel), а также с любыми программными продуктами, поддерживающими технологию OLE;
- богатый набор визуальных средств разработки, включающий средства автоматической, автоматизированной (мастера и шаблоны) и ручной разработки (конструкторы).

- хранение всех объектов, в том числе и программного кода, в одном файле;
- возможность публикации данных в сети Интернет;
- благодаря встроенному языку VBA в самом Access можно писать приложения, работающие с базами данных.

Любая СУБД должна выполнять ряд обязательных функций:
- определение структуры БД;
- заполнение БД;
- поиск и выборку данных;
- представление данных;
- поддержку языков БД;
- защиту физической и логической целостности данных;
- управление доступом к БД и др.

Основные элементы (объекты) MS Access включают в себя:
- таблицы (tables) для хранения данных;
- связи (relationships) между таблицами для обеспечения единства данных;
- формы (forms) для ввода и просмотра данных на экране;
- отчеты (reports) для вывода данных на печать;
- запросы (queries) для поиска необходимых данных по критерию;
- модули (modules) для хранения процедур на языке Visual Basic for Applications;
- макросы (macros) для автоматизации работы базы данных.
- страницы доступа к данным (data access pages)

2.2.2. Создание таблиц

После выбора подходящей для решаемой задачи СУБД необходимо перейти непосредственно к реализации БД в соответствии с ТЗ. Первым шагом будет преобразование сущностей и отношений в таблицы, а атрибутов сущностей – в поля таблиц.

Для создания таблиц потребуется перечень сущностей и отношений из ER-модели и перечень неповторяющихся данных (атрибутов сущностей).

При создании таблиц необходимо определить имена этих таблиц и параметры полей.

Для каждого поля необходимо:
- установить тип поля, размер и допустимый диапазон значений;
- определить вычислимость значений атрибута с использованием другой информации (если атрибут вычисляемый, то его необходимо исключить из таблицы);
- произвести внешнее кодирование, т. е. заменить длинные названия полей на короткие;
- создать механизмы повышения надежности ввода данных.

Все эти мероприятия служат одной важнейшей цели – снизить вероятность ошибок, допускаемых пользователями при вводе данных.

Источниками неверных данных в таблице могут быть ошибки действий (ошибки пользователя) и ошибки создания (ошибки разработчика).

Остановимся более подробно на ошибках действий. Пользователь может допускать ошибки при вводе данных в двух случаях: случайно (опечатки) и когда ему что-то непонятно. Таким образом, задачей разработчика является снижение вероятности таких ошибок до минимума.

Для этих целей в MS Access существует целый ряд инструментов, содержащихся в Конструкторе таблиц, которые будут перечислены ниже.

1. **Имя поля** – определяет, как следует обращаться к данным этого поля при автоматических операциях с базой (по умолчанию имена полей используются в качестве заголовков столбцов таблиц), является обязательным параметром поля и не может превышать 64 символа. При этом полное имя поля содержит в себе еще и имя таблицы (например: Студенты.Фамилия). Чем более понятно имя поля, тем меньше ошибок допустит пользователь при вводе данных (например, имя поля ПочтКод не является интуитивно понятным для пользователя в отличие от имени Почтовый индекс).

2. **Тип поля** – определяет тип данных, которые могут содержаться в данном поле, является обязательным свойством и в MS Access может принимать следующие значения (табл. 2.1).

Таблица 2.1

Тип поля	Описание	Размер
Текстовый	Для хранения обычного неформатированного текста ограниченного размера. Над числами в текстовом поле нельзя производить вычисления	До 255 символов
Поле Memo	Хранит длинный текст или числа, например пометки или описание. Не допускает регламентирования вводимых данных	До 65 535 символов
Числовой	Для хранения действительных чисел, над которыми можно производить математические вычисления	1, 2, 4 или 8 байт
Дата/время	Для хранения календарных дат и времени	8 байт
Денежный	Для хранения денежных сумм с денежной единицей (точность до 15 знаков в целой части и 4 в дробной)	8 байт
Счетчик	Специальный тип для уникальных (не повторяющихся в поле) натуральных чисел с автоматическим наращиванием или случайных чисел	4 байта
Логический	Тип для хранения логических данных, которые могут принимать только два значения (например, «Да/Нет», «Истина/Ложь», «Включено/Выключено»)	1 бит
Поле объекта OLE	Специальное поле, предназначенное для хранения объектов, созданных другими приложениями (например, документы Word, электронные таблицы Excel, рисунки, звуки). Объекты используют протокол OLE (Object Linking and Embedding) и	До 1 Гбайт

	могут быть связанными или внедренными	
Гиперссылка	Специальное поле для хранения адресов внешних объектов (URL, ссылки на документы Word и т. п.)	До 64 000 символов

Благодаря правильно выбранному типу поля снижается вероятность опечаток (так, например, в числовое поле нельзя случайно ввести букву вместо цифры).

3. **Размер поля** – определяет предельную длину данных, которые могут размещаться в данном поле. Для текстовых данных указывает максимальное количество символов, вводимых в поле. Для числовых данных размер поля – это поименованный диапазон значений (например, размер поля «байт» позволяет ввести целые числа от 0 до 255).

Размер поля также позволяет избежать случайных ошибок (например, при установке размера поля ПОЛ равным 1 пользователь не сможет ввести «муж», «мужской» и т. п.).

4. **Формат поля** – определяет способ форматирования данных в поле, позволяет улучшить восприятие данных (например, можно указать на необходимость отображения всех вводимых символов как прописные).

5. **Маска ввода** – регламентирует ввод каждого символа. Задает строку символов, каждый из которых кодирует обязательную или необязательную букву или цифру. Кроме этого, текстовые символы, прописанные в маске, будут автоматически отображаться (или храниться) в поле.

Например, маска ввода для поля Телефон может быть задана в виде (000) 000-0000 (0 – обязательная цифра). При этом в поле будут автоматически вводиться скобки и дефис.

6. **Подпись** – определяет заголовок столбца таблицы для данного поля. Подпись поля, по сути, является «псевдонимом» поля и может достигать 2048 символов. Имя поля должно быть коротким, так как при разработке БД использование длинных полей крайне неудобно. В то же время пользователю необходимо видеть длинные и понятные имена полей. Для этого используется подпись, заменяющая имя поля в таблицах, формах и отчетах. Если подпись не указана, то в качестве заголовка столбца используется Имя поля.

7. **Описание –** комментарий к полю, выводимый в строке состояния. Описание поля позволяет пользователю еще более подробно объяснить назначение поля и соответственно снизить вероятность ошибок.

8. **Значение по умолчанию –** то значение, которое вводится в ячейки поля автоматически.

В качестве значения по умолчанию имеет смысл использовать наиболее часто встречающееся значение. Это позволит существенно ускорить процесс ввода и уменьшить количество опечаток. Например, если в поле Страна по умолчанию задать «Россия», то для всех российских студентов в поле Страна это значение будет появляться автоматически.

9. **Условие на значение –** ограничение, используемое для проверки правильности ввода данных.

Существует условие на значение поля и записей. Условие на значение поля является логическим ограничением на вводимые данные в поле, поддерживающее непротиворечивость данных (например, Возраст > 18). Условие на значение записей позволяет сравнить данные, хранимые в разных полях (например: Год смерти > Год рождения).

10. **Сообщение об ошибке –** текстовое сообщение, которое выдается автоматически при попытке ввода в поле ошибочных данных (проверка выполняется автоматически, если задано свойство Условие на значение).

11. **Обязательное поле** – свойство, определяющее обязательность заполнения данного поля при наполнении базы.

12. **Пустые строки** – свойство, разрешающее ввод пустых строковых данных (значение Null).

13. **Индексированное поле** – поле, обладающее свойством, когда все операции, связанные с поиском или сортировкой записей по значению, хранящемуся в данном поле, существенно ускоряются. Кроме того, для индексированных полей можно сделать так, чтобы значения в записях проверялись по этому полю на наличие повторов, что позволяет автоматически исключить дублирование данных (уникальный индекс).

Важно отметить, что наличие или отсутствие вышеперечисленных свойств зависит от типа поля. Например, для полей, в которых хранятся рисунки, звукозаписи, видеоклипы и другие объекты OLE, большинство вышеуказанных свойств не имеют смысла.

В данном примере зададим следующие свойства полей:

Фамилия – текстовое, обязательное;

Имя – текстовое, обязательное;

Отчество – текстовое, обязательное;

Дата рождения – дата/время, обязательное;

Пол – текстовое, размер поля равен 1, обязательное;

Название факультета – текстовое, обязательное;

Название специальности – текстовое, обязательное;

Шифр специальности – текстовое, маска ввода – 000000 (шесть цифр), обязательное;

Оценка, полученная на экзамене – числовое, условие на значение >0 AND <30 (при 30-балльной шкале);

Дата экзамена – дата/время.

Таким образом, все требуемые атрибуты по таблицам распределены и заданы все необходимые параметры. Теперь необходимо реализовать заданные в ER-модели связи.

2.2.3. Связывание таблиц

Как уже было сказано в подразд. 1.4, для установки связей необходимо определить ключевые поля. Вернемся к нашему примеру. В данной БД будут следующие связи: Абитуриенты–Специальности (M:M), Абитуриенты–Экзамены (1:M), Специальности–Экзамены (1:M).

Для реализации этих связей в таблицах на стороне «один» необходимо определить первичные ключи. Поскольку в таблице Абитуриенты все поля могут повторяться, имеет смысл добавить новое поле Номер абитуриента и определить его в качестве первичного ключа.

В таблице Специальности имеется уникальное поле Шифр специальности. Зададим по нему первичный ключ.

Таблица Экзамены является отношением и должна хранить в себе внешние ключи Номер абитуриента и Шифр специальности. Добавим соответствующие поля в таблицу Экзамены.

Теперь можно установить связи, соединив в схеме данных соответствующие пары первичных и внешних ключей (рис. 6).

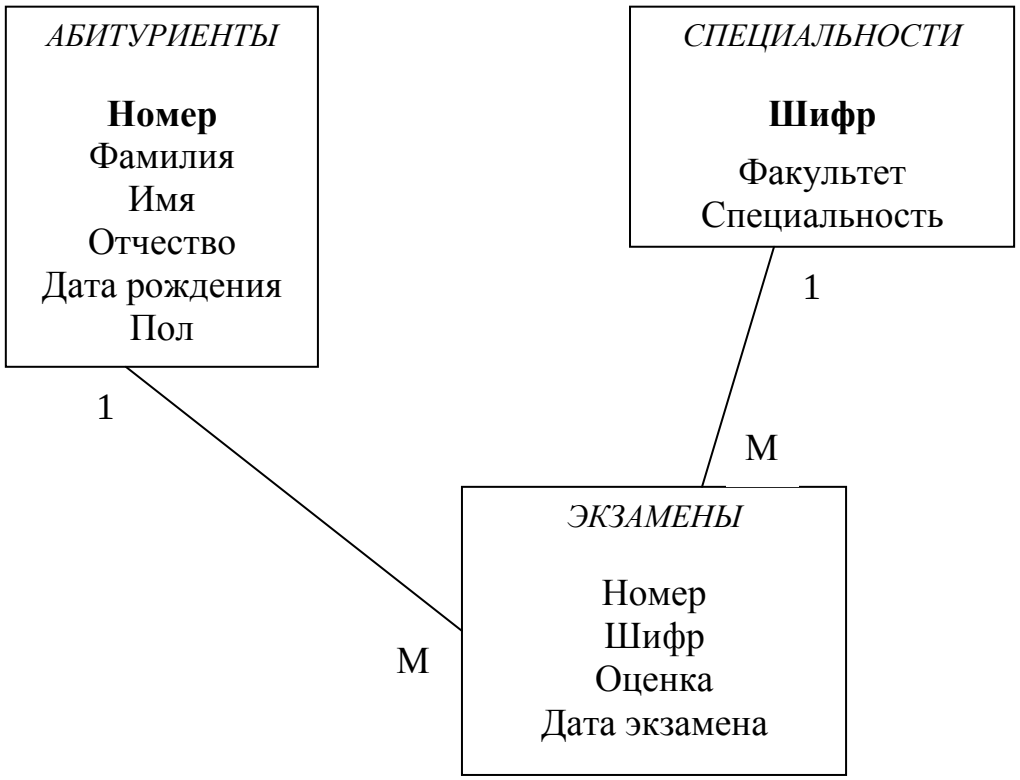

Рис. 6

Для соблюдения ссылочной целостности данных укажем на необходимость каскадного обеспечения целостности данных: каскадное обновление связанных полей и каскадное удаление связанных записей.

2.2.4. Разработка пользовательского интерфейса

Формы – это интерфейсный элемент, предназначенный для просмотра и ввода данных. Их цель – предоставить пользователю средства для заполнения только тех полей, которые ему заполнять положено. Одновременно с этим в форме можно разместить специальные элементы управления (счетчики, раскрывающиеся списки, переключатели, флажки и пр.) для автоматизации ввода. Источником формы может быть таблица (таблицы) или запрос (запросы). При этом происходит наследование свойств полей источника.

Формы позволяют пользователям вводить данные в таблицы БД без непосредственного доступа к самим таблицам. Это крайне важно по целому ряду причин.

Во-первых, для обеспечения конфиденциальности данных. Используя набор форм, можно эффективно разграничить доступ пользователей к данным в строгом соответствии с кругом персональных обязанностей. В банках, например, одни сотрудники имеют доступ к таблицам данных о клиентах, другие – к их расчетным счетам, третьи – к таблицам активов банка. Если и есть специальные службы, имеющие доступ ко всем информационным ресурсам банка (в целях контроля и анализа), то они лишены средств для внесения изменений. Все сделано так, чтобы один человек не мог совершить фиктивную операцию, независимо от того, какую должность он занимает.

Во-вторых, в целях безопасности: снижается риск того, что неумелыми действиями они повредят данные в таблицах.

В-третьих, в целях удобства ввода: на экране будут отображаться не все записи, а только одна.

Если структура БД хорошо продумана, то исполнители, работающие с базой, должны навсегда забыть о том, что в базе есть таблицы, а еще лучше, если они об этом вообще ничего не знают. Таблицы – слишком ценные объекты базы, чтобы с ними имел дело кто-либо, кроме разработчика.

Преимущества форм раскрываются особенно наглядно, когда происходит ввод данных с заполненных бланков. В этом случае форму делают графическими средствами так, чтобы она повторяла оформление бланка. Это заметно упрощает работу пользователя, снижает его утомление и предотвращает появление опечаток.

Ну и, наконец, в формах можно производить вычисления, создавая новые вычисляемые поля, что невозможно сделать в таблицах.

Создаваемые формы должны соответствовать функциям пользователей, определяемых на этапе проектирования. Так, в данном примере необходимо осуществлять ввод и просмотр данных об абитуриентах, экзаменах и специальностях.

По выполняемым функциям выделяют следующие виды форм:

- для управления транзакциями (их функции: добавление новых записей в таблицу и изменение существующих);
- для доступа к данным (их функции: представление информации для анализа – диаграммы, формы со статистическими сведениями);
- управляющие, или кнопочные (их функции: управление доступом к объектам базы данных).

По дизайну и структуре формы следующие:

- «в столбец» (отображает все поля одной записи, она удобна для ввода и редактирования данных);
- ленточная (отображает одновременно группу записей, ее удобно использовать для оформления вывода данных для их сравнения и анализа);

- табличная (по внешнему виду ничем не отличается от таблицы, на которой она основана, однако позволяет выводить только необходимые поля, в том числе и вычисляемые).

Форма имеет три основных раздела: область заголовка, область данных и область примечания. Разделы заголовка и примечания имеют чисто оформительское назначение, их содержимое напрямую не связано с таблицей или запросом, на котором основана форма. В этих разделах имеет смысл располагать название формы или итоговые вычисления (например, количество записей, суммарный доход, средний балл и т. д.). Раздел области данных имеет содержательное значение – в нем представлены элементы управления, при помощи которых выполняется отображение данных или их ввод.

Элементы управления бывают динамические и статические. Динамические элементы управления связаны с полями таблиц, а статические отображают свободные данные и хранятся внутри формы.

К динамическим элементам управления можно отнести:

- текстовые поля – служат для отображения данных из таблицы, ввода и редактирования данных, а также создания вычисляемых полей;
- флажки, переключатели, выключатели и их группы – служат для ввода и отображения данных из логических полей (например, отметка о зачете: да/нет);
- раскрывающиеся списки – позволяют осуществлять ввод посредством выбора значений из готового фиксированного списка;
- присоединенные рамки объектов – служат для отображения полей объектов OLE из таблицы (фотографий, рисунков, документов и т. д.)

К статическим элементам управления относят:

- надписи – произвольный текст;
- кнопки – служат для назначения им макросов или программ на VBA;
- вкладки – служат для создания многостраничных форм;
- линии, прямоугольники – для графического оформления формы;
- свободные рамки объектов – содержат внедренные в форму объекты (например, графическая эмблема компании). Для хранения рисунков можно использовать и элемент управления Рисунок.

Для создания вычисляемых полей в форме необходимо создать новое поле и разместить там соответствующее выражение. Для удобства создания формул можно использовать Построитель выражений.

Для анализа данных в формах применяют диаграммы. Диаграммы служат для наглядного графического представления информации, облегчая для пользователей сравнение и выявление тенденций и закономерностей в данных. При этом диаграмма может как сопровождать данные в форме, так и занимать всю форму. Для создания диаграмм используют приложение Microsoft Graph. Диаграммы могут быть глобальными (включающими все данные) или связанными с отдельной записью (отражающими данные только текущей записи и обновляющимися при переходе на другую запись).

Для создания кнопочных форм существует несколько путей, можно использовать:

1) элемент управления Кнопку для создания кнопок и назначения им макросов или процедур на языке VBA

2) мастер кнопок, позволяющий создать процедуру обработки события и назначить ее кнопке

3) диспетчер кнопочных форм для создания специальной таблицы (Switchboard Items), которая и хранит всю информацию о кнопочной форме.

Однако обычные формы, созданные на основе конкретной таблицы, не дают возможности одновременно видеть и вводить данные в связанные таблицы. Это крайне неудобно, так как не дает воспользоваться в полной мере всеми преимуществами установленных связей. Если необходимо обращаться к данным из разных таблиц, целесообразно создать **составные формы.**

При этом та форма, которая создана на основе таблицы со стороны «один», становится главной, а со стороны «много», – подчиненной. Встраивание подчиненной формы в главную можно осуществить в Конструкторе формы при помощи соответствующего элемента управления, методом Drag-and-Drop или при помощи Мастера форм.

В данном примере можно получить две составные формы: Студенты–Экзамены и Специальности–Экзамены.

Отчеты – это интерфейсный элемент, предназначенный для вывода данных на печать.

По своим свойствам и структуре отчеты во многом похожи на формы, но предназначены только для вывода данных, причем для вывода не на экран, а на печатающее устройство (принтер). В связи с этим отчеты отличаются тем, что в них приняты специальные меры для группирования выводимых данных и для вывода специальных элементов оформления, характерных для печатных документов (верхний и нижний колонтитулы, номера страниц, служебная информация о времени создания отчета и т. п.)

В отличие от форм отчеты:

– предназначены только для печати и не предназначены для вывода в окне;

- позволяют изменять значения исходных данных;

- имеют два режима работы: предварительный просмотр и режим конструктора.

Приемы создания и редактирования отчетов те же, что и для форм. Элементы управления в данном случае выполняют функции элементов оформления, поскольку печатный отчет – не интерактивный объект, в отличие от электронных форм.

Структура готового отчета отличается от структуры формы только увеличенным количеством разделов. Кроме разделов заголовка, примечания и данных, отчет может содержать разделы верхнего и нижнего колонтитулов. Если отчет занимает более одной страницы, эти разделы необходимы для печати служебной информации, например номеров страниц. Чем больше страниц занимает отчет, тем важнее роль данных, выводимых на печать через

эти разделы. Если для каких то полей отчета применена группировка, количество разделов отчета увеличивается, поскольку оформление заголовков групп выполняется в отдельных разделах.

Группировка записей может осуществляться как по полному значению (каждой категории соответствует уникальное значение), так и по диапазону значений (по первым знакам, временно́му или числовому интервалам). В случае необходимости подведения итогов не по отчету в целом, а по группам выражение, включающее статистическую функцию, размещают не в примечании отчета, а в примечании группы.

2.2.5. Создание запросов

Запросы служат для извлечения данных из таблиц и предоставления их пользователю в удобном виде. При помощи запросов выполняют такие операции, как отбор данных, их сортировку и фильтрацию, а также преобразование данных по заданному алгоритму, создание новых таблиц, автоматическое наполнение таблиц данными, импортированными из других источников, вычисления в таблицах и многое другое.

Особенность запросов состоит в том, что они черпают данные из базовых таблиц и создают на их основе временную результирующую таблицу. Основой запроса является критерий. Критерий – это совокупность условий, связанных логическими операторами (И, ИЛИ).

Для создания запросов используется язык SQL (Structured Query Language). Это универсальный компьютерный язык, применяемый для создания и модификации данных, а также управления ими в реляционных БД. SQL является информационно-логическим языком, а не языком программирования и основывается на реляционной алгебре.

Преимущества языка SQL:

- независимость от конкретной СУБД (несмотря на наличие диалектов и различий в синтаксисе, в большинстве своем тексты SQL-запросов могут быть достаточно легко перенесены из одной СУБД в другую);
- наличие стандартов и набора тестов (служит для выявления совместимости и соответствия конкретной реализации SQL общепринятому стандарту, что способствует «стабилизации» языка);
- декларативность (при помощи SQL программист описывает только, какие данные нужно извлечь или модифицировать, а каким образом это сделать, решает СУБД непосредственно при обработке SQL-запроса).

Основные (элементарные) инструкции SQL:

- SELECT (возвращает результирующий набор записей);
- FROM (определяет источник);
- WHERE (определяет выражения для условий отбора);
- ORDER BY (определяет поля сортировки).

Язык SQL был задуман как средство работы конечного пользователя, но со временем он стал настолько сложным, что превратился в инструмент программиста.

В связи с этим появился и другой способ создания запросов. **QBE** (Query by example) – способ создания запросов к БД с использованием образцов в виде текстовой строки, названия документа или списка документов. Система QBE преобразует пользовательский ввод в формальный запрос к БД, что позволяет пользователю делать сложные запросы, освобождая от необходимости изучать более сложные языки запросов, таких как SQL. В Access QBE реализован в виде Конструктора запросов.

Рассмотрим основные разновидности запросов.

1. Запрос на выборку.

Позволяет выбрать данные из полей таблиц, на основе которых запрос сформирован, удовлетворяющие определенному критерию. Например, в случае написания условия *Between 170 And 200* извлекутся все записи, для которых значения указанного поля находятся в указанном интервале.

Если необходимо, чтобы данные, отобранные в результате работы запроса на выборку, были упорядочены по какому-либо полю или сразу по нескольким полям, применяют сортировку по возрастанию или по убыванию.

В нижней части бланка запроса по образцу имеется строка Вывод на экран. По умолчанию предполагается, что все поля, включенные в запрос, должны выводиться на экран, но это не всегда целесообразно. Например, бывают случаи, когда некое поле необходимо включить в запрос, например потому, что оно является полем сортировки, но в то же время нежелательно, чтобы пользователь видел его содержание. В таких случаях отображение содержимого на экране подавляют сбросом флажка Вывод на экран. Примером может быть запрос на вывод списка сотрудников предприятия, отсортированный по количеству дней, пропущенных по болезни. Он позволит каждому оценить свое положение в этом списке, но не позволит точно узнать, кто и сколько дней болел.

При необходимости в запросе на выборку можно создавать новые вычисляемые поля. Так, если в таблице существуют поля Цена и Количество проданных товаров, то мы можем определить Стоимость заказа, создав новое поле с выражением *Стоимость: [Цена]*[Количество]*.

Иногда возникает ситуация, когда критерий запроса постоянно нуждается в изменении при сохранении его структуры. Например, необходимо регулярно узнавать успеваемость разных студентов. Поскольку пользователь сам изменять критерий запроса не может, разработчику придется создать запросы на все возможные фамилии, что невыполнимо. В этом случае может помочь запрос с параметрами. При выполнении такого запроса критерий изменяется самим пользователем при каждом запуске. Например, при записи в качестве условия *[Введите фамилию студента:]* пользователь увидит окно ввода с соответствующей надписью и введет интересующую его фамилию.

При необходимости над результатом выборки можно производить фильтрацию. Например, если необходимо определить пять самых высокооплачиваемых сотрудников, можно произвести в запросе сортировку по убыванию по полю Зарплата и указать на необходимость вывода пяти первых

значений (заменив на панели инструментов Набор значений со слова *Все* на число 5).

Если необходимо определить промежуточные итоги по группам записей, можно создать запрос с группировкой. Для этого используется режим группировки и статистические функции *Sum* (сумма), *Avg* (среднее), *Count* (количество), *Max* (максимум), *Min* (минимум). Например, для того чтобы подсчитать количество лиц женского пола, необходимо сделать группировку по Полу и использовать функцию Count по фамилии.

Перекрестные запросы очень близки сводным таблицам в Excel и позволяют создавать результирующие таблицы на основе результатов расчетов, полученных при анализе группы таблиц. Результаты статистических расчетов группируются по двум наборам данных, один из которых расположен в левом столбце таблицы, а второй – в верхней строке.

В некоторых случаях необходимо сохранить результат запроса в БД для последующей обработки. При этом используются запросы на создание таблицы. При повторном выполнении такого запроса старая таблица будет удалена, а на ее месте появится новая таблица с тем же именем.

2. Запрос на удаление.

Позволяет автоматизировать процесс удаления записей из одной или нескольких таблиц в соответствии с критерием. Запрос можно использовать, например, при отчислении студентов по результатам сессии.

3. Запрос на добавление.

Добавляет группу записей из одной или нескольких таблиц в конец одной или нескольких таблиц. Запрос может быть полезен, например, при переводе студентов, защитивших диплом из таблицы Студенты в таблицу Выпускники.

4. Запрос на обновление.

Данный тип запроса используется в тех случаях, когда необходимо выполнять отбор записей с последующим изменением для них значения определенного поля. Например, если нужно проиндексировать заработную плату сотрудников, можно делать это не вручную, меняя значения в поле Зарплата, а используя запрос на обновление.

В том случае, если необходимо извлечь данные сразу из нескольких таблиц используют **составные запросы.** Они отличаются от простых только тем, что в качестве источника запроса используется не одна, а несколько таблиц. При этом запросные связи, которые устанавливаются внутри данного запроса, могут отличаться от структурных связей в схеме данных.

При создании составных запросов необходимо правильно определять параметры объединения. Можно установить один из трех параметров:

1) объединение только тех записей, в которых связанные поля совпадают;

2) левое внешнее соединение (объединение всех записей со стороны «один» и только тех записей со стороны «много», в которых связанные поля совпадают);

3) правое внешнее соединение (объединение всех записей со стороны «много» и только тех записей со стороны «один», в которых связанные поля совпадают).

По умолчанию используется первый тип объединения.

Приведем пример. Пусть запросу необходимо извлечь информацию о фамилии, имени, отчестве и дате рождения абитуриента, а также об оценке, полученной им на экзамене. Добавив все требуемые поля в запрос, получим необходимую информацию. Однако если абитуриент подал документы, а экзамены еще не сдавал, то запрос не даст результата. Это объясняется тем, что в таблице Экзамены нет соответствующего внешнего ключа. Изменив параметр объединения на «левое внешнее», увидим всю информацию об абитуриенте, а вместо оценки – пустую ячейку.

Как уже было сказано ранее, для ускорения выполнения запросов рекомендуется использовать процедуру индексации полей, в которых часто происходит поиск. Записи в таблице представляют собой последовательную структуру и хранятся в том порядке, в каком осуществлялся ввод. В этом случае при выполнении запроса необходимо осуществлять поиск последовательным перебором всех значений, что занимает много времени.

Проблему можно решить, отсортировав все записи таблицы и перезаписав их в новом порядке. Но данную операцию пришлось бы делать каждый раз при добавлении и изменении записей в таблице, что значительно замедлило бы обработку информации.

Основной принцип индексации состоит в том, что от базовых таблиц никакой упорядоченности не требуется. Все записи в таблицы вносятся только в естественном порядке по мере их поступления, то есть в неупорядоченном виде. А для ускорения поиска используются индексные файлы. Индексный файл – это специальная служебная таблица. Она содержит значения атрибута, для которого создается индекс, и хранит ссылки на строки, в которых указано данное значение. Таким образом, получаем произвольный доступ ко всем записям (переходим от структуры последовательности к массивам).

Часто для ускорения процесса поиска той или иной записи необходимо создавать индекс на основании не одного, а нескольких полей таблицы одновременно, при этом индекс называется **составным.**

2.2.6. Использование средств автоматизации работы БД

В MS Access используются два инструмента для автоматизации работы с БД – макросы и модули. Эти категории объектов предназначены как для автоматизации повторяющихся операций при работе с системой управления базами данных, так и для создания новых функций путем программирования.

Макросы. В СУБД Microsoft Access макросы состоят из последовательности внутренних команд (макрокоманд) СУБД и являются одним из средств автоматизации работы с базой.

Макросы могут быть полезны для автоматизации часто выполняемых задач. Например, при нажатии пользователем кнопки можно запустить макрос, который распечатает отчет или откроет форму. В отличие от Word и Excel, в Access макросы создаются путем выбора последовательности стандартных

макрокоманд. Количество возможных макрокоманд невелико, около 50. В основном это открытие, переход, запуск.

Макрос может быть как собственно макросом, состоящим из последовательности макрокоманд, так и группой макросов. Если макросов много, объединение родственных макросов в группы может упростить управление базой данных.

В некоторых случаях требуется выполнять макрокоманду или серию макрокоманд только при выполнении некоторых условий. Например, если в макросе проверяется соответствие данных в форме условиям на значение, то для одних значений может потребоваться вывести одно сообщение, а для других значений – другое сообщение. В подобных случаях условия позволяют определить порядок передачи управления между макрокомандами в макросе.

Модули. Модуль – это набор описаний, инструкций и процедур на языке VBA (Visual Basic for Application), собранных в одну программную единицу и сохраненных под общим именем. Это одно из средств, при помощи которых разработчик базы может заложить в нее нестандартные функциональные возможности, удовлетворить специфические требования заказчика, повысить быстродействие системы управления, а также уровень ее защищенности.

Существуют два основных типа модулей – модули класса и стандартные модули.

К модулям класса относят модули форм и отчетов, которые связаны с определенной формой или отчетом. Они часто содержат процедуры обработки событий, запускаемые в ответ на событие в форме или отчете. Процедуры обработки событий используются для управления поведением формы или отчета и их откликом на события, такие как нажатие кнопки.

В 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.

Таким образом, создание пользовательского интерфейса является одним из важнейших этапов создания БД. Если интерфейс легко осваивается персоналом, прост в использовании, интуитивно понятен и устойчив к ошибкам, то пользователи легко научатся извлекать пользу из представленной в нем информации. В то же время если интерфейс лишен указанных качеств, то работа с такой системой неизбежно будет сопровождаться теми или иными проблемами.

После создания всех объектов происходит наполнение и тестирование БД. Затем она переходит в стадию эксплуатации.