Оглавление

Базы данных и файловые системы

Начнем с того, что с самого начала развития вычислительной техники образовались два основных направления ее использования. Первое направление - применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную. Становление этого направления способствовало интенсификации методов численного решения сложных математических задач, развитию класса языков программирования, ориентированных на удобную запись численных алгоритмов, становлению обратной связи с разработчиками новых архитектур ЭВМ.

Второе направление, которое непосредственно касается темы нашего курса, это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т.д.

На самом деле, второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными возможностями в части памяти. Понятно, что можно говорить о надежном и долговременном хранении информации только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. Оперативная память этим свойством обычно не обладает. В начале использовались два вида устройств внешней памяти: магнитные ленты и барабаны. При этом емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали последовательный доступ к данным. Магнитные же барабаны (они больше всего похожи на современные магнитные диски с фиксированными головками) давали возможность произвольного доступа к данными, но были ограниченного размера.

Области применения файлов

Прежде всего, конечно, файлы применяются для хранения текстовых данных: документов, текстов программ и т.д. Такие файлы обычно образуются и модифицируются с помощью различных текстовых редакторов. Структура текстовых файлов обычно очень проста: это либо последовательность записей, содержащих строки текста, либо последовательность байтов, среди которых встречаются специальные символы (например, символы конца строки).

Файловые системы обычно обеспечивают хранение слабо структурированной информации, оставляя дальнейшую структуризацию прикладным программам. В перечисленных выше случаях использования файлов это даже хорошо, потому что при разработке любой новой прикладной системы опираясь на простые, стандартные и сравнительно дешевые средства файловой системы можно реализовать те структуры хранения, которые наиболее естественно соответствуют специфике данной прикладной области.

Потребности информационных систем

Базы данных главным образом ориентированы на хранение, выбор и модификацию постоянно существующей информации. Структура информации зачастую очень сложна, и хотя структуры данных различны в разных информационных системах, между ними часто бывает много общего. На начальном этапе использования вычислительной техники для управления информацией проблемы структуризации данных решались индивидуально в каждой информационной системе. Производились необходимые надстройки над файловыми системами (библиотеки программ), подобно тому, как это делается в компиляторах, редакторах и т.д.

Но поскольку информационные системы требуют сложных структур данных, эти дополнительные индивидуальные средства управления данными являлись существенной частью информационных систем и практически повторялись от одной системы к другой. Стремление выделить и обобщить общую часть информационных систем, ответственную за управление сложно структурированными данными, явилось, на наш взгляд, первой побудительной причиной создания СУБД. Очень скоро стало понятно, что невозможно обойтись общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения данных.

Таким образом, СУБД решают множество проблем, которые затруднительно или вообще невозможно решить при использовании файловых систем. При этом существуют приложения, для которых вполне достаточно файлов; приложения, для которых необходимо решать, какой уровень работы с данными во внешней памяти для них требуется, и приложения, для которых безусловно нужны базы данных.

База данных

База данных (БД) — структурированный организованный набор данных, описывающих характеристики какой-либо физической или виртуальной системы.
База данных - совокупность взаимосвязанных данных, совместно хранимых в одном или нескольких компьютерных файлах.
«Базой данных» часто упрощённо или ошибочно называют системы управления базами данных **(СУБД)** — инструментальное программное обеспечение, предназначенное для организации ведения БД.
Виды концептуальных и логических моделей БД — сетевая модель, иерархическая модель, реляционная модель (ER-модель), многомерная модель, объектная модель.
На уровне физической модели электронная БД представляет собой файл или их набор в формате TXT, CSV, Excel, DBF, XML либо в специализированном формате конкретной СУБД. Также в СУБД в понятие физической модели включают специализированные виртуальные понятия, существующие в её рамках — таблица, табличное пространство, сегмент, куб, кластер и т.д.
В настоящее время наибольшее распространение получили реляционные базы данных. Картотеками пользовались до появления электронных баз данных. Сетевые и иерархические базы данных считаются устаревшими, объектно-ориентированные пока никак не стандартизированы и не получили широкого распространения. Некоторое возрождение получили иерархические базы данных в связи с появлением и распространением формата XML.

По виду модели БД разделяются:

Иерархические БД

В основе иерархических СУБД лежит довольно простая модель данных, которую можно представить себе в виде дерева ациклического ориентированного графа особого вида.

Дерево состоит из вершин, каждая из которых, кроме одной, имеет единственную родительскую вершину и несколько (в том числе ни одной) дочерних.

Вершина, не имеющая родительской, называется корнем дерева. Вершины, не имеющие дочерних, называются листьями. Остальные вершины являются ветвями.

Иерархические базы данных наиболее пригодны для моделирования структур, по своей природе являющихся иерархическими. В качестве примеров можно привести воинские подразделения или сложные механизмы, состоящие из более простых узлов, которые в свою очередь тоже можно подвергнуть декомпозиции.

Тем не менее существует значительное количество структур, не сводящихся к простой иерархии. Например, всем известное генеалогическое дерево, которое на самом деле не является деревом в строгом смысле, поскольку у большинства людей по два родителя. О более сложных структурах и говорить не приходится.

Иерархические СУБД быстро прошли пик популярности, которая обусловливалась их простотой в использовании и ранним появлением на рынке, когда основные конкуренты еще не дозрели для коммерческого использования. Затем их многочисленные недостатки сделали их неконкурентоспособными, и в настоящее время иерархическая модель представляет исключительно исторический интерес.

Сетевые СУБД

Подобно иерархической, сетевую модель также можно представить себе в виде ориентированного графа. Но в этом случае граф может содержать циклы, т.е. вершина может иметь несколько родительских.

Такая структура намного гибче и выразительнее предыдущей и пригодна для моделирования гораздо более широкого класса задач. В этой модели вершины представляют собой сущности, а соединяющие их ребра отношения между ними.

Сетевые СУБД имели гораздо больший успех и долго господствовали на рынке СУБД. В немалой степени их успеху способствовала энергичная деятельность Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL). Эта организация тщательно проработала спецификации сетевой модели и ее архитектуру, что позволило создать ряд успешных коммерческих продуктов, не последнее место среди которых занимал некогда весьма популярный COBOL.

70-е годы XX века фактически стали эпохой расцвета сетевой модели. Сетевые СУБД весьма прочно укрепились на рынке, и реляционной модели пришлось с боем завоевывать свое место под солнцем. В истории информатики навечно останется Великий Спор, который на самом деле явился решающим сражением сетевой и реляционной моделей. В рядах сторонников сетевой архитектуры был сам великий Чарльз Бахман, и только гений Эдгара Кодда позволил реляционной модели одержать победу.

Реляционные СУБД

Реляционные СУБД являются в настоящий момент самыми распространенными. Их реализации существуют на всех мало-мальски пригодных для этого платформах (от персональных компьютеров до мэйнфреймов), для всех операционных систем и для всех применений от

простейших продуктов, предназначенных для ведения картотек индивидуального пользования, до сложнейших распределенных многопользовательских систем.

Несмотря на такое пестрое разнообразие, все эти СУБД имеют в основе общую основу реляционную модель данных, разработанную Коддом в 70-х годах XX столетия. С виду эта модель довольно проста: база данных выглядит как простой набор взаимосвязанных таблиц. Но за внешней простотой кроется мощный и вместе с тем изящный математический аппарат реляционной алгебры, которая в свою очередь базируется на целом ряде математических дисциплин, среди которых логика, исчисление предикатов, теория множеств

Немалую роль в успехе реляционных СУБД играет также язык SQL, разработанный специально для запросов к реляционным БД. Это достаточно простой и в то же время выразительный язык, при помощи которого можно выполнять достаточно изощренные запросы к базе.

Разумеется, предшествующие СУБД также имели языки описания данных (ЯОД) и языки манипулирования данными (ЯМД). SQL объединил в себе обе эти функции. Но самой привлекательной его особенностью, особенно для пользователей-непрофессионалов в программировании, является то, что можно строить запросы на основе непроцедурного подмножества SQL. Это означает, что в формулировке запроса указывается, что должно содержаться в результате, а не как его получить. Имеются, правда, и процедурные элементы языка, например, операторы организации ветвления и циклов, но их применения зачастую удается избежать. При работе же с сетевыми БД программист был вынужден использовать навигационные процедуры, отвлекаясь при этом от решения самой задачи.

Нормализация таблиц

Существуют определенные правила, которые должны соблюдаться при создании базы данных. Пользователь должен пройти следующие этапы, которые называют "нормализацией":

удалить повторяющиеся группы атрибутов. Группой считаются два и более атрибута, их необходимо вынести в отдельную таблицу.

удалить неключевые атрибуты и те, которые не имеют отношения к первичному ключу или относятся только к части первичного ключа. Если первичный ключ составной, то любой атрибут таблицы должен быть связан с обоими частями ключа.

описать каждую таблицу: каждому атрибуту дать имя и указать вид информации, который используется в таблице (числовой, дата, денежный и т.п.). Название атрибута должно быть уникальным в рамках одной таблицы, обычно названия пишут латинскими буквами без пробелов.

работа с пользовательским интерфейсом, обозначение обязательных и необязательных для заполнения полей.

забота о безопасности, если это необходимо: шифрование данных, ограничение прав доступа.

Объектно-ориентированные базы данных, в которой данные оформлены в виде моделей объектов, включающих прикладные программы, которые управляются внешними событиями.
В наиболее общей и классической постановке объектно-ориентированный подход базируется на концепциях:
объекта и идентификатора объекта;
атрибутов и методов;
классов;
иерархии и наследования классов.

Многомерные Программное обеспечение OLAP используется при обработке данных из различных источников. Эти программные продукты позволяют реализовать множество различных представлений данных и характеризуются тремя основными чертами: многомерное представление данных; сложные вычисления над данными; вычисления, связанные с изменением данных во времени.

База данных (БД) — это организованная структура, предназначенная для хранения информации.

Функции СУБД

Традиционных возможностей файловых систем оказывается недостаточно для построения даже простых информационных систем. Мы выявили несколько потребностей, которые не покрываются возможностями систем управления файлами: поддержание логически согласованного набора файлов; обеспечение языка манипулирования данными; восстановление информации после разного рода сбоев; реально параллельная работа нескольких пользователей. Можно считать, что если прикладная информационная система опирается на некоторую систему управления данными, обладающую этими свойствами, то эта система управления данными является системой управления базами данных (СУБД).

Основные функции СУБД

1. Непосредственное управление данными во внешней памяти

Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для убыстрения

доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но подчеркнем, что в развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

2. Управление буферами оперативной памяти

СУБД обычно работают с БД значительного размера; по крайней мере этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

Заметим, что существует отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предположении, что в будущем объем оперативной памяти компьютеров будет настолько велик, что позволит не беспокоиться о буферизации. Пока эти работы находятся в стадии исследований.

3. Управление транзакциями

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Таким образом, поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД (если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД.

То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД (на самом деле, это несколько идеализированное представление, поскольку в некоторых случаях пользователи многопользовательских СУБД могут ощутить присутствие своих коллег).

4. Журнализация

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД. Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журнализуются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального

журнала. В некоторых СУБД так и делают, но в большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу).
Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после жесткого сбоя является достаточно длительным.

5. Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные. Мы рассмотрим более подробно языки ранних СУБД в следующей лекции.
В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language)
Прежде всего**, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными**. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.
Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.
Специальные операторы языка SQL позволяют определять так называемые **представления БД, фактически являющиеся хранимыми в БД запросами** (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

6. Авторизация доступа к объектам БД

Производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

Реляционная модель данных

Понятие реляционный (англ. Relation - отношение) связано с разработками известного американского специалиста в области баз данных Е. Кодда.
Эти модели характеризуются простотой структуры данных, удобным для пользователя табличным представлением и возможностью использования формального аппарата алгебры отношений и реляционного исчисления для обработки данных.

Свойства таблицы в реляционной модели данных

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

- каждый элемент таблицы - один элемент данных;

- все столбцы в таблице однородные, т.е. все элементы в столбце имеют одинаковый тип (числовой, символьной и т.д.) и длину;
- каждый столбец имеет уникальное имя;

Введение в базы данных Microsoft Access

СУБД Access входит в состав Microsoft Office и предназначена для работы с реляционными БД, т.е. представленными в табличной форме. В отличие от табличного процессора Excel, Access имеет более развитые средства для отбора данных из взаимосвязанных таблиц, формирования новых таблиц и отчетов.
Характерной особенностью баз данных, созданных в Access, является хранение создаваемых таблиц и средств для обработки данных в одном файле, имеющем расширение .mdb.
Достоинством Access является возможность создания СУБД (т.е. программы управления) без программирования. Однако, для сложных СУБД применение программирования на встроенном языке Visual Basic for Applications (VBA) позволяет повысить эффективность системы управления.
Основные объекты окна БД имеют следующее назначение:

- таблица — основное средство для хранения информации в БД;
- запрос — это инструмент для извлечения необходимой информации из исходных таблиц и представления ее в удобной форме.
- форма — это основное средство для ввода данных, управления СУБД и вывода результатов на экран монитора;
- отчет — это специальное средство для формирования выходных документов и вывода их на принтер;
- макросы в Access представляют собой совокупность внутренних команд, предназначенных для автоматизации работы с БД;
- модули являются программами, создаваемыми средствами языка VBA, и похожи на макросы в Word и Excel.

Таблицы и запросы связываются между собой с помощью схемы данных.
Таблицы, запросы, формы и отчеты БД можно создавать в двух режимах: вручную с помощью конструктора или при помощи Мастера. Выбор средства определяется конкретными обстоятельствами, однако следует заметить, что мастер быстро создает заготовку объекта, которую обычно требуется "дорабатывать" вручную.

Технология разработки СУБД содержит несколько этапов, основными из которых являются:

- проектирование структуры БД и связей между таблицами;
- разработка структуры отдельных таблиц и ввод данных в таблицы;
- разработка запросов;
- разработка схемы данных, реализующей запроектированные связи между таблицами и запросами;
- разработка макросов и программных модулей для управления БД;
- разработка форм для реализации интерфейса управления БД;
- разработка отчетов для печати документов.

Приведенная последовательность этапов не является жесткой. Обычно разработчику СУБД приходится многократно возвращаться к одним и тем же этапам, постепенно уточняя проект.

Свойства таблицы

Основным элементом БД является таблица. Столбцы таблицы БД называются *полями*, а строки — *записями.* Первым этапом создания таблицы БД является задание ее структуры, т.е. определение количества и типа полей. Вторым этапом является ввод и редактирование записей в таблицу*. БД считается созданной, даже если она пустая.*
Поля таблицы просто определяют ее структуру и групповые свойства данных, записываемых в ячейках. Рассмотрим основные свойства полей БД.
Имя поля — определяет как надо обращаться к данным поля (имена используются как заголовки таблиц). Каждое поле в таблице должно иметь уникальное имя, удовлетворяющее соглашениям об именах объектов в Access. Оно является комбинацией из букв, цифр, пробелов и специальных символов, за исключением символов " . " " ! " " ' " " [" "] " . Имя не может начинаться с пробела и содержать управляющие символы с кодами ASCII от 00 до 31. Максимальная длина имени 64 символа.
Тип данных (Data Type). Тип данных определяется значениями, которые предполагается вводить в поле, и операциями, которые будут выполняется с этими значениями. В Access допускается использование девяти типов данных. Список возможных типов данных вызывается нажатием кнопки списка при выборе типа данных каждого поля:

Текстовый (Text) - тип данных по умолчанию. Текст или цифры, не участвующие в расчетах. Число символов в поле не должно превышать 255. Максимальное число символов, которое можно ввести в поле, задается в свойстве Размер поля (FieldSize). Пустые символы в неиспользуемой части поля не сохраняются
Поле МЕМО (Memo). Длительный текст, например, некоторое описание или примечание. Максимальная длина 64 000 символов
Числовой (Number). Числовые данные, используемые в математических вычислениях. Конкретные варианты числового типа и их длина задаются в свойстве Размер поля (FieldSize). Для проведения денежных расчетов определен другой тип данных - Денежный (Currency)
Денежный (Currency). Денежные значения и числовые данные, используемые в расчетах, проводящихся с точностью до 15 знаков в целой и до 4 знаков в дробной части. Длина поля 8 байт. При обработке числовых значений из денежных полей выполняются вычисления с фиксированной точкой более быстрые, чем вычисления для полей с плавающей точкой, кроме того, при вычислениях предотвращается округление. Учитывая эти обстоятельства, рекомендуется для полей, в которых планируется хранить числовые значения с указанной точностью, использовать денежный тип данных
Дата/время (Data/Time). Значения даты или времени, относящиеся к годам с 100 по 9999 включительно. Длина поля 8 байт
Счетчик (AutoNumber). Тип данных поля, в которое для каждой новой записи автоматически вводятся уникальные целые, последовательно возрастающие (на 1), или случайные числа. Значения этого поля нельзя изменить или удалить. Длина поля 4 байта для длинного целого, для кода репликации - 128 байт. По умолчанию в поле вводятся последовательные значения. В таблице не может быть более одного поля этого типа. Используется для определения уникального ключа таблицы.
Логический (Yes/No). Логические данные, которые могут иметь одно из двух возможных значений Да/Нет; Истина/Ложь; Вкл./Выкл. (Yes/No; True/False; On/Off). Длина поля 1 бит
Поле объекта OLE (OLE Object). Объект (например, электронная таблица Microsoft Exce1, документ Microsoft Word, рисунок; звукозапись или другие данные в двоичном формате), связанный или внедренный в таблицу Access. Длина поля - до 1 Гигабайта (ограничивается объемом диска). Для полей типа OLE и MEMO не допускается сортировка и индексирование Гиперссылка (Hyperlink). В качестве гиперссылки можно указывать путь к файлу на жестком диске, путь UNC или адрес URL. Если щелкнуть мышью на поле гиперссылки, Access выполнит переход на соответствующий объект, документ, страницу Web или другое место назначения. Максимальная длина 64 000 символов
Мастер подстановок...(Lookup Wizard...). Выбор этого типа данных запускает мастера подстановок. Мастер строит для поля список значений на основе полей из другой таблицы. Значения в такое поле будут вводиться из одного из полей списка. Соответственно, фактически тип данных поля определяется типом данных поля списка. Возможно также определение поля со списком постоянных значений.
Общие свойства поля
Общие свойства задаются для каждого поля на вкладке Общие (General) и зависят от выбранного типа данных. Для отображения свойств поля необходимо установить курсор на строке соответствующего поля:

Размер поля (FieldSize) задает максимальный размер данных, сохраняемых в поле. Для поля с типом данных Текстовый задается размер от 1 до 255 байтов (по умолчанию 50 байт)
Для поля с типом данных Счетчик можно задать:
Длинное Целое (Long Integer) - 4 байта
Код репликации (Replication ID) - 128 байт
Для поля с типом данных Числовой можно задать:
Байт (Byte) для целых чисел от 0 до 255, длина поля 1 байт
Целое (Integer) для целых чисел от -32.768 до + 32.767, занимает 2 байта
Длинное целое (Long Integer) для целых чисел от -2.147.483.648 до +2.147.483.647, занимает 4 байта
Дробные с плавающей точкой 4 байта (Single) для чисел от -3,4x1038 до + 3,4xЮ38 с точностью до 7 знаков
Дробные с плавающей точкой 8 байт (Double) для чисел от -1,797xЮ308 до +1,797xЮ308 с точностью до 15 знаков
Действительное (Decimal) для целых чисел от -1038-1 до 1038-1 (при рабо-те с проектами, которые хранятся в файлах типа .abp) и от -1028-1 до 1028-1 (.mdb) с точностью до 28 знаков, занимает 12 байт
Код репликации- Глобальный уникальный идентификатор (Globally unique identifier, GUID), занимает 16 байт. Поля такого типа используются Access для создания системных уникальных идентификаторов реплик, наборов реплик, таблиц, записей и других объектов при репликации баз данных.

Рекомендуется задавать минимально допустимый размер поля, который понадобится для сохраняемых значений, так как сохранение таких полей требует меньше памяти, и обработка данных меньшего размера выполняется быстрее. Изменение размера поля с большего на меньший в таблице, имеющей данные, может привести к их искажению или полной потере.

Изменения в данных, которые происходят вследствие изменения свойства Размер поля, нельзя отменить после их сохранения в конструкторе таблиц.

Формат поля (Format) является форматом отображения заданного типа данных и задает правила представления данных при выводе их на экран или печать.

В Access определены встроенные стандартные форматы отображения для полей с типами данных Числовой (Number), Дата/время (Date/Time), Логический (Yes/No) и Денежный (Currency). Ряд этих форматов совпадает с настройкой национальных форматов, определяемых в окне Язык и стандарты панели управления Microsoft Windows. Пользователь может создать собственный формат для всех типов данных, кроме OLE, с помощью символов форматирования.

Для указания конкретного формата отображения необходимо выбрать в раскрывающемся списке одно из значений свойства Формат поля (Format). Формат поля используется для отображения данных в режиме таблицы, а также применяется в форме или отчете при отображении этих полей.

Число десятичных знаков (DecimalPlaces) задает для числового и денежного типов данных число знаков после запятой. Можно задать число от О до 15. По умолчанию (значение Авто, Autto) это число определяется установкой в свойстве Формат поля (Format). Следует иметь в виду, что установка этого свойства не действует, если свойство Формат поля (Format) не установлено или выбрано значение Основной (General Number). Свойство Число десятичных знаков (Decimal Places) влияет только на количество десятичных знаков, отображаемых на экране, и не влияет на число сохраняемых десятичных знаков. Для изменения числа сохраняемых знаков: нужно изменить свойство Размер поля (FieldSize)

Маска ввода - определяет шаблон для ввода данных. Например, можно установить шаблон для ввода даты: **.**.****.

Подпись (Caption) поля задает текст, который выводится в таблицах, формах, отчетах.

Значение по умолчанию - содержит значение, устанавливаемое по умолчанию в данном поле таблицы. Например, если в поле Город ввести значение по умолчанию Тюмень, то при вводе записей, это поле можно пропускать, а соответствующее значение будет введено автоматически

Условие на значение (ValidationRule) позволяет осуществлять контроль ввода, задает ограничения на вводимые значения, при нарушении условий запрещает ввод и выводит текст, заданный свойством Сообщение об ошибке (ValidationText)

Обязательное поле - установка, указывающая на то, что данное поле требует обязательного заполнения для каждой записи.
Пустые строки - установка, которая определяет, допускается ли ввод в данное поле пустых строк ("").
Индексированное поле - определяет простые индексы для ускорения поиска записей.

Сообщение об ошибке (ValidationText) задает текст сообщения, выводимый на экран при нарушении ограничений, заданных свойством Условие на значение (ValidationRule)

Тип элемента управления
На вкладке Подстановка (Lookup) в окне конструкторЗначение по умолчанию - содержит значение, устанавливаемое по умолчанию в данном поле таблицы. Например, если в поле Город ввести значение по умолчанию Тюмень, то при вводе записей, это поле можно пропускать, а соответствующее значение будет введено автоматическиа таблиц задается свойство Тип элемента управления (DisplayControl). Это свойство определяет, будет ли отображаться поле в таблице и в форме в виде Поля (Text Box), Списка (List Box) или Поля со списком (Combo Box). Таким образом определяется вид элемента управления, используемого по умолчанию для отображения поля.
Если для поля выбран тип элемента управления Список (List Box) или Поле со списком (Combo Box), на вкладке Подстановка (Lookup) появляются дополнительные свойства, которые определяют источник данных для строк списка и ряд других характеристик списка. В качестве

источника данных для списка выбирается таблица, с которой осуществляется постоянная связь, что обеспечивает актуальное состояние списка.

Типы таблиц и ключей в реляционных базах данных

Реляционные базы данных характеризуются наличием некоторых типов таблиц и ключей, позволяющих определить отношения между таблицами. Для того чтобы понять принципы разработки реляционных баз данных, требуется дать определения различных типов реляционных ключей и таблиц:

Базовая таблица. В реляционной базе данных базовой таблицей называется таблица, которая включает один или несколько столбцов свойств объекта и содержит первичный ключ, который однозначно определяет этот объект. Более того, базовая таблица должна содержать первичный ключ. Базовые таблицы часто называют первичными, поскольку они имеют первичный ключ.

Промежуточная таблица. Таблица, не являющаяся базовой (т. к. она не объединяет свойства объекта или не содержит поле первичного ключа), которая используется для обеспечения связей между другими таблицами, называется таблицей отношений. Ключевые поля в таблицах отношений должны быть внешними ключами, связанными с первичными ключами базовой таблицы. Проще говоря, таблица отношений состоит только из внешних ключей и не содержит независимых элементов данных.

Определение первичного ключа

Каждая таблица в реляционной базе данных должна иметь уникальный (первичный) ключ, который может быть простым или составным, включающим несколько полей (до 10). Для определения ключа выделяются поля, составляющие ключ, и на панели инструментов Конструктор таблиц (Table Design) нажимается кнопка Ключевое поле (Primary Key) или выполняется команда меню Правка| Ключевое поле (Edit| Primary Key).

Первичный ключ. Первичный ключ состоит из набора значений, которые однозначно определяют запись базовой таблицы. Любому значению первичного ключа должна соответствовать одна и только одна строка таблицы. Первичный ключ включает одно поле только в том случае, если это поле не содержит повторяющихся значений.

Составные ключи. Если для выполнения условий, накладываемых на значения первичного ключа, заданный ключ включает несколько полей таблицы, то тогда он называется составным.

Внешние ключи. Внешний ключ — это столбец, значения которого соответствуют значениям первичного ключа другой связанной таблицы.

Индексы

В большинстве реляционных баз данных реализация ключей требует создания специальных объектов базы данных, именуемых индексами. Индекс (index) представляет собой список позиций записей, который показывает порядок их следования. Записи в реляционной таблице могут быть неупорядоченными. Тем не менее, любая запись имеет физическую позицию (положение) в файле базы данных. Эта позиция может меняться СУБД в процессе обработки данных, однако, в каждый конкретный момент времени она уникальна для каждой записи. Физическое положение (позиция) записи может измениться в процессе вставки, обновления или удаления данных, а также при выполнении некоторых манипуляций, осуществляемых сервером базы данных (таких, как компрессия файлов базы данных). Однако большинство современных настольных СУБД и все СУБД типа клиент/сервер поддерживают сопровождаемые индексы (maintained indexes), т. е. индексы меняются при изменении положения записей и значений полей. В этом случае любая вставка, обновление или удаление записей приводит к тому, что производится запись самой таблицы и всех индексов, связанных с этой таблицей. Из-за этого увеличивается время, необходимое для модификации данных.

Необходимо отметить, что некоторые типы полей, например тето-и BLOB-ПОЛЯ, не могут быть проиндексированы.

Для ключевого поля автоматически строится индекс. В этом можно убедиться, просмотрев информацию об индексах таблицы. Окно Индексы (Indexes) вызывается щелчком на кнопке просмотра и редактирования индексов Индексы (Indexes) на панели инструментов или выполнением команды меню Вид| Индексы (View| Indexes).

В этом окне индексу первичного ключа присвоено имя PrimaryKey, в столбце Поле (FieldName) перечисляются имена полей, составляющие индекс. Индекс ключевого поля всегда уникален и не допускает пустых полей в записях. Если первичный ключ не установлен пользователем до сохранения вновь созданной таблицы, Access спросит о необходимости создания первичного ключа. При ответе "Да" Access создаст первичный ключ с типом данных Счетчик (AuttoNumber).

Взаимосвязи таблиц

При создании в Access схемы данных в ней определяются и запоминаются связи между таблицами. Это позволяет системе автоматически использовать связи, один раз определенные в схеме данных, при создании форм, запросов, отчетов на основе взаимосвязанных таблиц, а пользователь освобождается от необходимости указывать эти связи при конструировании этих объектов. Схема данных базы графически отображается в своем окне, где таблицы представлены списками полей, а связи - линиями между полями разных таблиц.

Создание схемы данных

Создание схемы данных начинается в окне Базы данных (Database) с выполнения команды Сервис|Схема данных (Tools|Relationships) или нажатия кнопки Схема данных (Relationships) на панели инструментов базы данных.

Одно-многозначные (1:M) или одно-однозначные (1:1) связи. Схема данных прежде всего ориентирована на работу с таблицами, отвечающими требованиям нормализации, между которыми могут быть установлены одно-многозначные (1:M) или одно-однозначные (1:1) связи, для которых может автоматически поддерживаться связная целостность. Поэтому схему данных целесообразно строить в соответствии с информационно-логической моделью.

При построении схемы данных Access автоматически определяет по выбранному полю связи тип отношения между таблицами. Если поле, по которому нужно установить связь, является уникальным ключом как в одной таблице, так и в другой, Access выявляет отношение один-к-одному. Если поле связи является уникальным ключом в одной таблице (главной таблицы связи), а в другой таблице (подчиненной таблице связи) является не ключевым или входит в составной ключ, то есть значения его могут повторяться, Access выявляет отношение один-ко-многим между записями главной таблицы к подчиненной. В этом случае можно задать автоматическое поддержание целостности связей.

Отношение многие-ко-многим. Отношение многие-ко-многим предполагает, что каждой записи в одной таблице соответствует несколько записей в другой. При этом каждая сторона отношения выглядит как отношение один-ко-многим.

Однако если рассматривать взаимосвязь таблиц с двух сторон, становится очевидным, что ни одна из таблиц не может быть главной и для их связывания необходима третья таблица.

Связующая таблица представляет собой промежуточную таблицу, которая служит мостом между двумя таблицами в отношении многие-ко-многим. Ее ключ состоит из ключевых полей этих таблиц, с каждой из которых она связана отношением один-ко-многим. Помимо ключевых полей, связующая таблица должна содержать хотя бы одно поле, которого нет в связываемых таблицах, но которое имеет значение для каждой из них. Таким образом, отношение многие-ко-многим складывается из отношений многие-к-одному и один-ко-многим.

Связи-объединения. Между двумя таблицами может быть установлена связь-объединение по некоторому полю связи. Для связи-объединения может быть выбран один из трех способов объединения записей:

Способ 1 *- объединение только тех записей, в которых связанные поля обеих таблиц совпадают (производится по умолчанию);*

Способ 2 *- объединение тех записей, в которых связанные поля обеих таблиц совпадают, а также объединение всех записей из первой таблицы, для которых нет связанных во второй, с пустой записью второй таблицы;*

Способ 3 *- объединение тех записей, в которых связанные поля обеих таблиц совпадают, а также объединение всех записей из второй таблицы, для которых нет связанных в первой, с пустой записью первой таблицы.*

Такой тип связи может быть определен, если связь характеризуется отношением 1:1 или 1:M, а также если тип отношения не может быть определен системой, то есть если не выполняются условия для этих отношений. Например, при выборе в главной таблице в качестве поля связи неключевого поля или поля, входящего в составной ключ, Access сообщает, что тип отношения не может быть определен. В этом случае между таблицами возможно установление только связи-объединения.

Связь-объединение обеспечивает объединение записей таблиц, имеющих одинаковые значения в поле связи. Причем производится объединение каждой записи из одной таблицы с каждой записью из другой таблицы при условии равенства значений в поле связи. Кроме того, если выбран второй или третий вариант в результат объединения могут быть добавлены записи из таблицы, для которых нет логически связанных записей в другой таблице. Последние два варианта часто необходимы при решении практических задач. *Примером такой задачи может*

быть формирование записей студентов с результатами успеваемости как в случае полученной оценки по предмету, так и при отсутствии оценки. При отсутствии оценки соответствующее поле будет пустым.

Создание связей между таблицами. При определении связей в схеме данных удобно использовать информационно-логическую модель в каноническом виде, по которой легко определить главную и подчиненную таблицу каждой одно-многозначной связи, поскольку в такой модели главные объекты всегда размещены выше подчиненных. Эти связи являются основными в реляционных базах данных, т. к. одно-однозначные связи используются лишь в редких случаях, когда приходится разделять большое количество полей, определяемых одним и тем же ключом, по разным таблицам, имеющим разный регламент обслуживания.

Устанавливая в окне схемы данных связи типа 1:М между парой таблиц, надо выделить в главной таблице уникальное ключевое поле, по которому устанавливается связь. Далее, при нажатой кнопке мыши, протащить курсор в соответствующее поле подчиненной таблицы.

Обеспечение целостности данных
При создании схемы данных пользователь включает в неё таблицы и устанавливает связи между ними. Для связей типа 1:1 и 1:М можно задать параметр обеспечения связной целостности данных, а также автоматическое каскадное обновление и удаление связанных записей.

Обеспечение связной целостности данных означает, что Access при корректировке базы данных обеспечивает для связанных таблиц контроль за соблюдением следующих условий:
В подчиненную таблицу не может быть добавлена запись с несуществующим в главной таблице значением ключа связи;
В главной таблице нельзя удалить запись, если не удалены связанные с ней записи в подчиненной таблице;
Изменение значений ключа связи в записи главной таблицы невозможно, если в подчиненной таблице имеются связанные с ней записи.
При попытке пользователя нарушить эти условия в операциях добавления и удаления записей или обновления ключевых данных в связанных таблицах Access выводит соответствующее сообщение и не допускает выполнения операции.

Установление между двумя таблицами связи типа 1:М или 1:1 и задание для нее параметров **целостности** данных возможно только при следующих **условиях**:

- Связываемые поля имеют одинаковый тип данных, причем имена полей могут быть различными;
- Обе таблицы сохраняются в одной базе данных Access;
- Главная таблица связывается с подчиненной по первичному простому или составному ключу (уникальному индексу) главной таблицы.

Access автоматически отслеживает целостность связей при добавлении и удалении записей и изменении значений ключевых полей, если между таблицами в схеме данных установлена связь с параметрами обеспечения целостности. При действиях, нарушающих целостность связей таблиц, выводится сообщение**. Access не позволяет установить параметр целостности для связи таблиц, если ранее введенные в таблицы данные не отвечают требованиям целостности**.

Каскадное обновление и удаление связанных записей
Если для выбранной связи обеспечивается поддержание целостности, можно задать режим каскадного обновления связанных полей и режим каскадного удаления связанных записей.
В режиме каскадного обновления связанных полей при изменении значения поля связи в записи главной таблицы, Access автоматически изменит значения в соответствующем поле в подчиненных записях.

В режиме каскадного удаления связанных записей при удалении записи из главной таблицы будут автоматически удаляться все связанные записи в подчиненных таблицах. При удалении записи из главной таблицы выполняется каскадное удаление подчиненных записей на всех уровнях, если этот режим задан на каждом уровне.

При удалении записей непосредственно в таблице или через форму выводится предупреждение о возможности удаления связанных записей.