**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Поволжский государственный университет телекоммуникаций и информатики»

**КОЛЛЕДЖ СВЯЗИ**

|  |  |
| --- | --- |
|  |  |

**Учебное пособие по дисциплине**

**ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ**

для специальностей: 09.02.04 – Информационные системы

09.02.03 - Программирование в компьютерных системах

Составила

преподаватель Малбасарова Г. Х.

Самара

2015

|  |  |
| --- | --- |
| Рассмотрено на заседании П(Ц)К  «Информационные системы и технологии»  Протокол № 5 от 12.01.2015г.  Председатель П(Ц)К \_\_\_\_\_\_\_\_\_\_Шомас Е.А. | Утверждаю  Зам. директора по УВР  \_\_\_\_\_\_\_\_\_Логвинов А.В.  «12» января 2015г. |

Основы проектирования баз данных. Учебное пособие. Составлено преподавателем КС ПГУТИ Малбасаровой Г.Х., Самара КС ПГУТИ, 2015г. – 3 п.л.

**СОДЕРЖАНИЕ**

**Раздел 1. Основы теории проектирования баз данных**

**Тема 1.1.**Определение и назначение баз данных. Типы взаимосвязей в моделях баз данных

**Тема 1.2.** Нормализация баз данных. Средства доступа к данным

**Тема 1.3.** Этапы проектирования баз данных. Системы управления базами данных

**Раздел 2. Использование СУБД ACCESS для создания баз данных**

**Тема 2.1.** Основные характеристики и возможности СУБД Access

**Тема 2.2.** Создание таблиц в СУБД Access

**Тема 2.3.** Индексирование: понятие индекса, типы индексных файлов. Создание, активация и удаление индекса. Переиндексирование

**Тема 2.4.** Обработка данных в базе

**Раздел 3. Организация запросов SQL**

**Тема 3.1.** Исторические аспекты. Структура и типы данных языка SQL

**Тема 3.2.** Операторы языка SQL

**Раздел 4. Организации интерфейса с пользователем**

**Тема 4.1.** Понятие объекта, свойства и характеристики объекта. Создание экранной формы: свойства, события и методы

**Тема 4.2.** Хранимые процедуры и триггеры. Обеспечение достоверности, целостности и непротиворечивости данных. Каскадные воздействия. Макросы

**Тема 4.3.** Формирование и вывод отчетов

**РАЗДЕЛ 1. ОСНОВЫ ТЕОРИИ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ**

**ТЕМА 1.1 ОПРЕДЕЛЕНИЕ И НАЗНАЧЕНИЕ БАЗ ДАННЫХ. ТИПЫ ВЗАИМОСВЯЗЕЙ В МОДЕЛЯХ БАЗ ДАННЫХ**

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.

**Второе направление** 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, пре доставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет довольно сложную структуру. Классическими примера ми информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т.д.

Второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными возможностями. Надежное и долговременное хранение информации возможно только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. Оперативная память этим свойством обычно не обладает. Используемые в ранних ЭВМ два вида устройств внешней памяти — магнитные ленты и барабаны — были несовершенными. Магнитные ленты обладали достаточно большой емкостью, но по своей физической природе обеспечивали лишь последовательный доступ к данным. Магнитные барабаны, обеспечивая возможность произвольного доступа к данным, имели ограниченный размер. Появление новых носителей данных — в первую очередь, жестких дисков — дало толчок к работам по созданию информационных компьютерных систем.

Основу любой информационной системы составляет база данных, т. е. набор данных, организованных специальным образом.

«**База данных** — это объективная форма представления и организации совокупности данных (например, статей, расчетов), систематизированных таким образом, чтобы эти данные могли быть найдены и обработаны с помощью ЭВМ».

**Файл** — это место фактического хранения информации. В файле различают структуру и собственно данные. Структура файла остается неизменной, а информация (данные) может изменяться при операциях обращения к нему.

В качестве основной структурообразующей единицы хранимых в файле данных принимается хранимая запись. Хранимые записи состоят из фиксированной совокупности полей, служащих для представления значений какого-либо типа (чисел, литерных строк, дат, булевских значений, денежных единиц и т.д.), и могут иметь формат фиксированной или переменной длины. Полям, как правило, присваиваются уникальные в данной базе имена, ассоциируемые с предметной областью. Если в качестве примера базы данных рассмотреть картотеку сотрудников некоторого абстрактного предприятия, то единицей хранимых данных может быть запись персональной информации по каждому сотруднику с полями: табельный номер (формат поля — целое число); фамилия, имя, отчество (формат поля — литерная строка определенной длины); дата рождения (формат поля — дата); заработная плата (формат поля — действительное число) и т.д.

Информационные системы ориентированы главным образом 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.

Поскольку информационные системы содержат сложные структуры данных, дополнительные индивидуальные средства управления этими данными, являясь существенной частью информационных систем, практически повторялись от одной системы к другой. Стремление выделить общую часть информационных систем, ответственную за управление сложно структурированными данными, явилось первой побудительной причиной создания систем управления базами данных.

**Компоненты наиболее полного варианта СУБД следующие:**

• среда пользователя, дающая возможность непосредственного управления данными с клавиатуры;

• алгоритмический язык для программирования прикладных систем обработки данных, реализованный как интерпретатор (последний позволяет быстро создавать и отлаживать программы);

• компилятор для придания завершенной программе вида готового коммерческого продукта в форме независимого ЕХЕ-файла;

• программы-утилиты для быстрого программирования рутинных операций (генераторы отчетов, форм, таблиц, экранов, меню и других приложений).

Собственно СУБД это инструментальная оболочка пользователя, а ввиду того, что такая среда ориентирована на немедленное удовлетворение запросов пользователя, это всегда система-интерпретатор. Наличие в СУБд языка программирования позволяет создавать сложные системы обработки данных, ориентированные на конкретные задачи и конкретного пользователя.

**Области применения баз данных**

Автоматизированные информационные системы (АИС), основу которых составляют базы данных, появились в 60-х годах ХХ века в военной промышленности и бизнесе — там, где были накоплены значительные объемы полезных данных. Первоначально АИС были ориентированы лишь на работу с информацией фактического характера — числовыми или текстовыми характеристиками объектов. Затем по мере развития техники появилась возможность обработки текстовой информации на естественном языке.

Принципы хранения разных видов информации в АИС аналогичны, но алгоритмы ее обработки определяются характером ин формационных ресурсов. Соответственно различают два класса АИС: документальные и фактографические.

Документальные АИС служат для работы с документами на естественном языке. Наиболее распространенный тип документальных АИС — информационно-поисковые системы, предназначенные для накопления и подбора документов, удовлетворяющих заданным критериям. Эти системы могут выполнять просмотр и подборку монографий, публикаций в периодике, сообщений пресс- агентств, текстов законодательных актов и т.д.

Фактографические АИС оперируют фактическими сведениями, представленными в формализованном виде, и используются для решения задач обработки данных.

Обработка данных — специальный класс решаемых на ЭВМ задач, связанных с вводом, хранением, сортировкой, отбором и группировкой записей данных однородной структуры. К задачам этого класса относятся: учет товаров в магазинах и на складах; начисление зарплаты; управление производством, финансами, телекоммуникациями и т. п.

Различают фактографические АИС оперативной обработки данных, подразумевающие быстрое обслуживание относительно простых запросов от большого числа пользователей, и фактографические АИС аналитической обработки, ориентированные на выполнение сложных запросов, требующих проведения статистической обработки исторических (накопленных за некоторый промежуток времени) данных, моделирования процессов предметной области и прогнозирования развития этих процессов.

Таким образом, АИС применяются в следующих областях:

• организация хранилищ данных;

• системы анализа данных;

• системы принятия решений;

• мобильные и персональные базы данных;

• географические базы данных;

• мультимедиа базы данных;

• распределенные информационные системы.

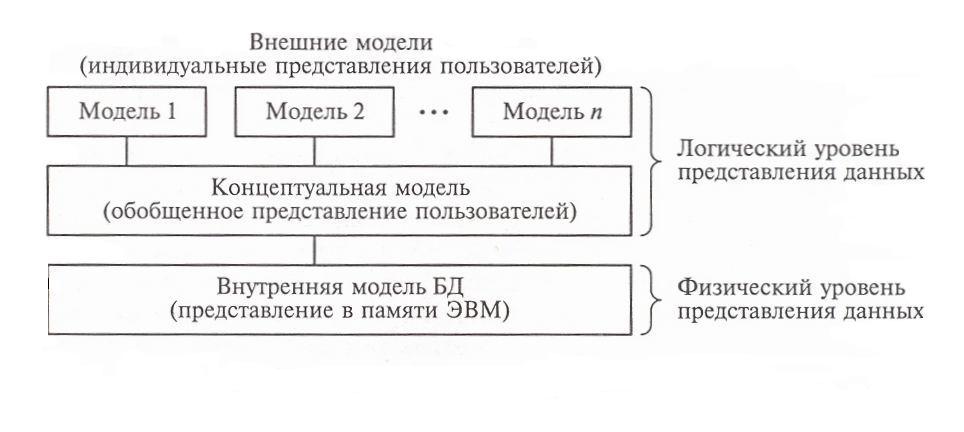
Каждая информационная система в зависимости от назначения имеет дело с той или иной частью конкретного мира, которую принято называть ее предметной областью. Анализ предмет ной области является необходимым начальным этапом разработки любой информационной системы. Именно на этом этапе определяются информационные потребности всей совокупности пользователей будущей системы, которые, в свою очередь, предопределяют содержание ее базы данных. Предметная область конкретной информационной системы рассматривается, прежде все го, как некоторая совокупность реальных объектов, которые представляют интерес для ее пользователей. Примерами объектов предметной области могут служить персональные ЭВМ, программные продукты и их пользователи. Каждый из этих объектов обладает определенным набором свойств (атрибутов). Так, например, компьютер характеризуется названием фирмы-производителя, идёнтификатором модели, типом микропроцессора, объемом оперативной и внешней памяти, типом графической карты и т. д.

Информационный объект это описание некоторой сущности предметной области, т. е. реального объекта, процесса, явления или события. Информационный объект (сущность) образуется совокупностью логически взаимосвязанных атрибутов (свойств), представляющих собой качественные и количественные характеристики объекта (сущности).

Между объектами предметной области могут существовать связи, имеющие различный содержательный смысл. Эти связи могут быть обязательными или факультативными (необязательными).

Если вновь порожденный объект оказывается по необходимости связанным с каким-либо объектом предметной области, то между этими двумя объектами существует обязательная связь. В противном случае связь является факультативной. Например, обязательная связь Замещает существует между двумя объектами СОТРУДНИК и ДОЛЖНОСТЬ в предметной области кадровой информационной системы, т. е. каждый принимаемый в организацию сотрудник зачисляется на какую-либо должность и не может быть сотрудника, не замещающего какой-либо должности. В то же время связь Замещает между типами объектов СОТРУДНИК и ДОЛЖНОСТЬ является факультативной, поскольку могут существовать вакантные должности. Совокупность объектов предметной области и связей между ними характеризует структуру предметной области. Множество объектов предметной области, значения атрибутов объектов и связи между ними могут изменяться во времени. Изменения могут сводиться к появлению новых или исключению из рассмотрения некоторых существующих объектов в предметной области, установлению новых или разрушению существующих связей между ними. Следовательно, с каждым моментом времени можно сопоставить некоторое состояние предметной области.

Информационно-логическая модель (ИЛМ) — это совокупность информационных объектов (сущностей) предметной области и связей между ними. Процесс создания информационной модели начинается с определения концептуальных требований будущих пользователей БД. Требования отдельных пользователей интегрируются в едином обобщенном представлении, которое называют концептуальной моделью данной предметной области.



Такая модель отображает предметную область в виде взаимосвязанных объектов без указания способов их физического хранения.

Концептуальная модель представляет собой интегрированные концептуальные требования всех пользователей к базе данных данной предметной области. При этом усилия разработчика должны быть направлены в основном на структуризацию данных, принадлежащих будущим пользователям БД и выявление взаимосвязей между ними.

Возможно, что отраженные в концептуальной модели взаимосвязи между объектами окажутся впоследствии нереализуемыми средствами выбранной СУБд, что потребует ее изменения. Вер сия концептуальной модели, которая может быть реализована конкретной СУБд, называется логической моделью.

Логическая модель, отражающая логические связи между атрибутами объектов вне зависимости от их содержания и среды хранения, может быть реляционной, иерархической или сете вой. Таким образом, логическая модель отображает логические связи между информационными данными в данной концептуальной модели.

Различным пользователям в информационной модели соответствуют различные подмножества ее логической модели, которые называются **внешними моделями пользователей**.

Таким образом, внешняя модель пользователя представляет собой отображение его концептуальных требований в логической модели и соответствует тем представлениям, которые этот пользователь получает о предметной области на основе логической модели. Следовательно, насколько хорошо спроектирована внешняя модель, настолько полно и точно информационная мо ель отображает предметную область и настолько полно и точно работает автоматизированная система управления этой предмет ной областью.

Логическая модель отображается в физическую память, которая может быть построена на электронных, магнитных, оптических, биологических или других принципах.

Внутренняя модель предметной области определяет размещение данных, методы доступа к ним и технику индексирования в данной логической модели и иначе называется **физической моделью.**

Информационные данные любого пользователя в БД должны быть независимы от всех других пользователей, т. е. не должны оказывать влияния на существующие внешние модели. Это положение отражает первый уровень независимости данных. С другой стороны, внешние модели пользователей никак не связаны с типом физической памяти, в которой будут храниться данные, и с физическими методами доступа к этим данным. Это положение отражает второй уровень независимости данных.

**Три типа логических моделей баз данных**

Ядром любой базы данных является модель данных. Модель данных — это совокупность структур данных и операций их обработки.

По способу установления связей между данными различают **иерархическую**, **сетевую** и **реляционную модели**.

**Иерархическая модель** позволяет строить базы данных с древо видной структурой, где каждый узел содержит свой тип данных (сущность). На верхнем уровне дерева в этой модели имеется один узел — корень, на следующем уровне располагаются узлы, связанные с этим корнем, затем узлы, связанные с узлами предыдущего уровня и т. д. При этом каждый узел может иметь только одного предка.

Компьютеры

Прочие устройства

Сотрудники

Оборудование

Начальник

Отделы

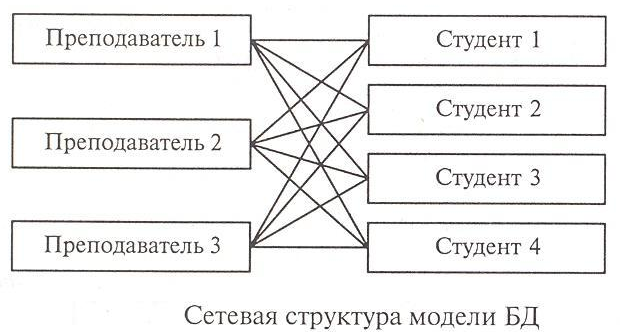
Филиалы

Организация

Иерархическая древовидная структура модели БД

Поиск данных в иерархической системе всегда начинается с корня. Затем производится спуск с одного уровня дерева на другой, пока не будет достигнут искомый уровень. Перемещения по системе от одной записи к другой осуществляются с помощью ссылок.

**Основные достоинства иерархической модели** — простота описания иерархических структур реального мира и быстрое выполнение запросов. Однако не всегда удобно каждый раз начинать поиск нужных данных с корня, а другого способа перемещения по базе в иерархических структурах нет. Указанный недостаток снят в сетевой модели, где (по крайней мере, теоретически) возможны связи всех информационных объектов со всеми.



В данной модели каждый преподаватель может обучать многих (теоретически всех) студентов и каждый студент может обучаться у многих (теоретически у всех) преподавателей. Поскольку на практике это, естественно, невозможно, приходится прибегать к некоторым ограничениям. Использование иерархической и сетевой моделей ускоряет доступ к информации в базе данных. Однако, поскольку каждый элемент данных должен содержать ссылки на некоторые другие элементы, требуются значительные ресурсы как дисковой, так и основной памяти ЭВМ. Недостаточность основной памяти, конечно, снижает скорость обработки данных. Кроме того, для таких моделей характерна сложность реализации системы управления базами данных.

Реляционная модель была разработана в начале 70-х годов ХХ в. Коддом. Простота и гибкость этой модели привлекли к ней внимание разработчиков, и уже 80-х годах ХХ в. она получила широкое распространение. Таким образом, реляционные СУБд оказались промышленным стандартом.

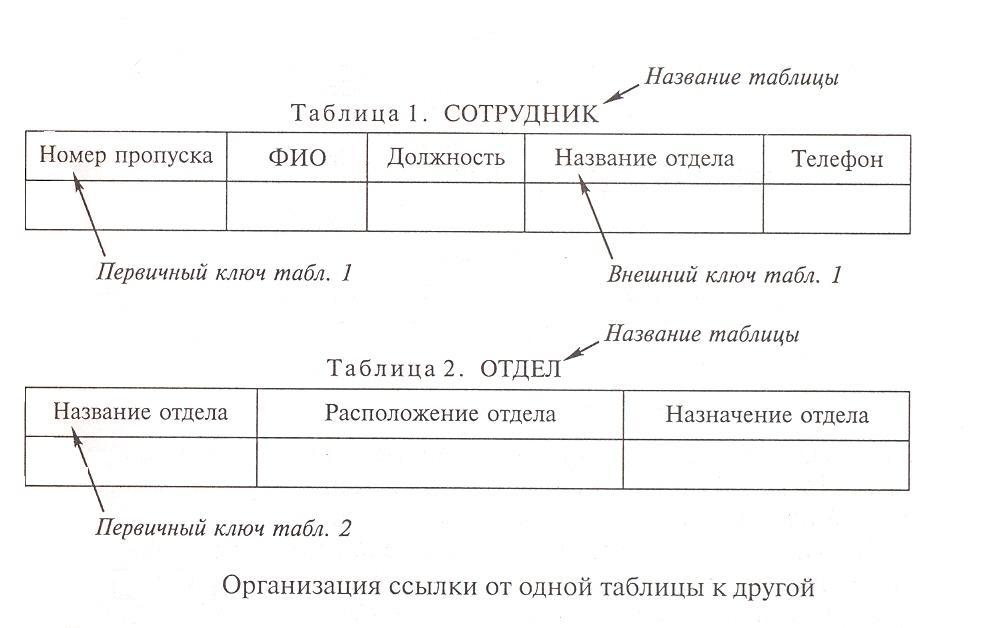
Реляционная модель опирается на систему понятий реляционной алгебры, важнейшими из которых являются таблица, строка, столбец, отношение и первичный ключ, а все операции в этом случае сводятся к манипуляциям с таблицами. В реляционной модели информация представляется в виде прямоугольных таблиц, каждая из которых состоит из строк и столбцов и имеет имя, уникальное внутри базы данных.

Таблица отражает объект реального мира — сущность, а каждая ее строка (запись) отражает один конкретный экземпляр объекта — экземпляр сущности. Каждый столбец таблицы имеет уникальное для данной таблицы имя. Располагаются столбцы в соответствии с порядком следования их имен, принятом при создании таблицы.

В отличие от столбцов строки не имеют имен, порядок их следования в таблице не определен, а число — логически не ограничено. Так как строки в таблице не упорядочены, невозможно выбрать строку по ее позиции. Номер, имеющийся в файле у каждой строки, не характеризует ее, так как его значение изменяется при удалении строк из таблицы. Логически не существует первой и последней строк.

Реляционные системы исключили необходимость сложной навигации, поскольку данные представлены в них не в виде одного файла, а независимыми наборами, и для отбора данных используются операции реляционной алгебры — прикладной теории множеств.

В каждой таблице реляционной модели должен быть столбец (или совокупность столбцов), значение которого однозначно идентифицирует каждую ее строку. Этот столбец (или совокупность столбцов) и называется первичным ключом таблицы.



Если таблица удовлетворяет требованию уникальности первичного ключа, она называется отношением. В реляционной модели все таблицы должны быть преобразованы в отношения. Отношения реляционной модели связаны между собой. Связи поддерживаются внешними ключами.

**Внешний ключ** — это столбец (совокупность столбцов), значение которого однозначно характеризует значения первичного ключа другого отношения (таблицы).

Говорят, что отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором та же совокупность столбцов является первичным ключом.

В приведенном примере отношение СОТРУДНИК ссылается на отношение ОТДЕЛ через название отдела.

Схема реляционной таблицы (отношения) представляет собой совокупность имен полей, образующих ее запись:

НАЗВАНИЕ ТАБЛИЦЫ (Поле 1, Поле 2, ..., Поле п).

Например, для таблиц, показанных на рисунке, имеем следующие схемы (курсивом выделены первичные ключи):

СОТРУДНИК (Номер пропуска, ФИО, Должность, Название отдела, Телефон);

ОТДЕЛ (Название отдела, Расположение отдела, Назначение отдела).

Объектно-ориентированная модель баз данных начала разрабатываться в связи с появлением объектно-ориентированных язы ков программирования в 90-е годы ХХ века. Такого рода базы хранят методы классов, а иногда и постоянные объекты классов, что позволяет осуществлять беспрепятственную интеграцию между данными и их обработкой в приложениях.

Доминирование реляционной модели в современных СУБД определяется:

наличием развитой теории (реляционной алгебры);

наличием аппарата сведения других моделей данных к реляционной модели;

наличием специальных средств ускоренного доступа к информации;

наличием стандартизированного высокоуровневого языка запросов к БД, позволяющего манипулировать ими без знания конкретной физической организации БД во внешней памяти.

**ТИПЫ ВЗАИМОСВЯЗЕЙ В МОДЕЛИ**

На практике часто используются связи, устанавливающие различные виды соответствия между объектами «связанных» типов, — это один к одному (1:1), один ко многим (1:М), многие ко многим (М: М).

Связь один к одному означает, что каждому экземпляру первого объекта (А) соответствует только один экземпляр второго объекта (В) и, наоборот, каждому экземпляру второго объекта (В) соответствует только один экземпляр первого объекта (А).

Связь один ко многим означает, что каждому экземпляру одного объекта (А) может соответствовать несколько экземпляров другого объекта (В), а каждому экземпляру второго объекта (В) может соответствовать только один экземпляр первого объекта (А).

Связь многие ко многим означает, что каждому экземпляру одного объекта (А) могут соответствовать несколько экземпляров второго объекта (В) и, наоборот, каждому экземпляру второго объекта (В) могут соответствовать тоже несколько экземпляров первого объекта (А).

**Пример.** Рассмотрим совокупность следующих информационных объектов:

СТУДЕНТ (Номер студента, ФИО, Дата рождения, Номер группы);

СТИПЕНДИЯ (Номер студента, Размер стипендии);

ГРУППА (Номер группы, Специальность);

ПРЕПОДАВАТЕЛЬ (Код преподавателя, ФИО, Должность).

Здесь информационные объекты СТУДЕНТ и СТИПЕНДИЯ связаны отношением один к одному, так как каждый студент может иметь только одну стипендию и каждая стипендия может быть назначена только одному студенту.

Информационные объекты ГРУППА и СТУДЕНТ связаны отношением один ко многим, так как одна группа может включать в себя много студентов, в то время как каждый студент может обучаться только в одной группе.

Информационные объекты СТУДЕНТ и ПРЕПОДАВАТЕЛЬ связаны отношением многие ко многим, так как один студент может обучаться у многих преподавателей и один преподаватель может обучать многих студентов.

**Обеспечение непротиворечивости и целостности данных в базе**

Для пользователей АИС важно, чтобы база данных отображала предметную область однозначно и непротиворечиво, т. е. чтобы она удовлетворяла условию целостности.

**Выделяют два основных типа ограничений по условию целостности данных в базе.**

**1.** Каждая строка таблицы должна отличаться от остальных ее строк значением хотя бы одного столбца.

**Пример.** Сотрудники одного отдела могут оказаться полными тезками, иметь одинаковые должность и телефон. Наличие в таблице столбца Номер пропуска превращает ее в отношение. Таким образом, первое ограничение по условию целостности данных в базе обеспечивается наличием в таблице-отношении первичного ключа.

**2.** Внешний ключ не может быть указателем на несуществующую строку той таблицы, на которую он ссылается. Это ограничение называется ограничением целостности данных в базе по ссылкам.

**Пример**. В столбце Название отдела таблицы СОТРУДНИК хранятся сведения о принадлежности сотрудников к отделу, т. е. этот столбец является внешним ключом для ссылки на таблицу ОТДЕЛ для обеспечения ограничения целостности данных по ссылкам каждое название отдела из таблицы СОТРУДНИК должно принадлежать конкретному столбцу из таблицы ОТДЕЛ.

В реальных базах данных названия не делают ключевыми из-за их длины, замедляющей процесс поиска, и возможности изменения, создающей сложности с сопровождением системы.

**Основы реляционной алгебры**

Поскольку каждая таблица в реляционной БД является отношением, действия над ними базируются на операциях реляционной алгебры. Исключение составляют лишь операции создания и заполнения таблиц данными (присваивания), а также операции описания и переименования столбцов таблицы.

В теории реляционной алгебры отношение рассматривается как множество, строки таблицы называются кортежами, столбцы — атрибутами. Над отношениями выполняются традиционные операции теории множеств.

Ограничение отношения (выборка) создание нового от ношения отбором в него строк отношения-операнда (исходного отношения), которые удовлетворяют условию ограничения.

Проекция отношения — создание нового отношения отбором в него определенных столбцов отношения—операнда.

Объединение отношений — создание нового отношения, со держащего все кортежи отношений-операндов. При этом операнды должны иметь одинаковые атрибуты.

**Пример.** Объединить поступающие из цехов отчеты о выпуске но вой продукции за прошедший месяц, содержащие следующие данные:

номер цеха, код продукции, дату выпуска и количество выпущенной продукции, с данными общей таблицы ВЫПУСК ПРОДУКЦИИ, имеющей ту же структуру. Для этого к кортежам

ВЫПУСК ПРОДУКЦИИ (Номер цеха, Код продукции, Дата выпуска, Количество)

добавляют кортежи

НОВАЯ ПРОДУКЦИЯ (Номер цеха, Код продукции, Дата выпуска, Количество).

Поскольку атрибуты приведенных операндов совпадают, таблица НОВАЯ ПРОДУКЦИЯ объединяется с исходной.

Пересечение отношений — создание нового отношения, содержащего строки, общие для сравниваемых операндов. При этом операнды должны иметь одинаковые атрибуты.

**Пример.** Рассмотрим пересечение отношений с выполнением операций ограничения и проекции.

Для экзаменационных ведомостей нужной группы сначала выполним ограничение исходных отношений, отобрав из каждого из них в новое отношение кортежи, удовлетворяющие следующему условию:

Оценка = Отлично.

В результате получим списки отличников группы по дисциплинам.

Теперь выполним проекцию полученных отношений, отобрав из каждого из них только атрибуты Номер зачетной книжки и ФИО студента. Получим новые списки отличников, в которых остались только номера зачетных книжек и фамилии студентов.

Таким образом, получим искомое отношение — СПИСОК ОТЛИЧНИКОВ, содержащее номера зачетных книжек и фамилии, общие для всех списков отличников.

**Пример.** Подготовить сведения о выпуске новых видов продукции за последний квартал.

Для решения этой задачи выполняем ограничение отношения ВЫПУСК ПРОДУКЦИИ по следующему условию: дата выпуска меньше последней даты прошлого квартала.

Результат такого ограничения поместим в исходную таблицу.

Затем выполним следующее ограничение для исходной таблицы: дата выпуска меньше первой даты прошлого квартала. Полученный результат занесем в конечную таблицу.

Разность отношений исходной и конечной таблиц даст искомые сведения.

Произведение отношений — создание нового отношения, в котором имеются все атрибуты 1-го и 2-го операндов, а строки получены по парным сцеплением строк их отношений. Число кортежей — мощность нового отношения — равна произведению мощностей 1-го и 2-го отношений. При этом множества атрибутов от ношений не должны пересекаться.

Произведение отношений используется при решении задач подбора пар из двух множеств, например поставщиков и потребителей. Для этого сначала составляют все возможные пары, а затем по конкретному критерию отбирают из них подходящие.

**Пример**. По двум заданным отношениям требуется найти произведение. деление отношений — создание нового отношения, содержащего атрибуты 1-го операнда, отсутствующие во 2-м операнде, и кортежи 1-го операнда, которые совпали с кортежами 2-го операнда. Для выполнения этой операции 2-й операнд должен содержать лишь атрибуты, совпадающие с атрибутами 1 -го.

**Пример**. Требуется отобрать студентов группы, получающих стипендию, используя список, содержащий следующие сведения: ФИО, дата рождения, шифр группы и признак наличия стипендии (да, нет).

Для решения задачи создадим вспомогательное отношение с атрибутами Шифр группы и Признак наличия стипендии. Затем заполним один кортеж этого отношения, поместив в него шифр заданной группы и отметку о получении стипендии (да).

В результате деления исходного списка на вспомогательное отношение получим искомый список с атрибутами ФИО и Дата рождения.

Соединение отношений — создание нового отношения, кортеж которого является результатом сцепления кортежей операндов (исходных отношений).

Различают соединения отношений двух видов: естественное и по условию.

При соединении отношений по условию производятся сцепление строк их операндов и проверка полученной строки на соответствие заданному условию. Если условие выполнено, то полученная строка включается в результирующее отношение.

При естественном соединении отношений производятся сцепление строк их операндов и включение полученной строки в результирующее отношение без проверки. Такое соединение используют, когда отношения-операнды обладают общими атрибутами.

Одни и те же данные могут группироваться в таблицы различными способами. Группировка атрибутов в отношениях должна быть рациональной, т. е. минимизирующей дублирование данных и упрощающей процедуры их обработки и обновления. Устранение избыточности данных, являющееся одной из важнейших задач при проектировании баз данных, обеспечивается нормализацией.

**Контрольные вопросы**

1. Назовите виды моделей баз данных. Отличия, достоинства, недостатки
2. Назовите виды взаимосвязей между объектами баз данных. Отличия, достоинства, недостатки
3. Что такое первичный ключ
4. Что такое внешний ключ, его отличие от первичного

**ТЕМА 1.2. НОРМАЛИЗАЦИЯ БАЗ ДАННЫХ. СРЕДСТВА ДОСТУПА**

**Нормализация** — это формальный аппарат ограничений на формирование таблиц (отношений), который позволяет устранить дублирование, обеспечивает непротиворечивость хранимых данных и уменьшает трудозатраты на ведение (ввод, корректировку) базы данных.

Процесс нормализации заключается в разложении (декомпозиции) исходных отношений БД на более простые отношения. При этом на каждой ступени этого процесса схемы отношений приводятся в нормальные формы, для каждой ступени нормализации имеются наборы ограничений, которым должны удовлетворять отношения БД. Тем самым удаляется из таблиц базы избыточная не ключевая информация.

Процесс нормализации основан на понятии функциональной зависимости атрибутов: атрибут А зависит от атрибута В (В —> А), если в любой момент времени каждому значению атрибута В соответствует не более одного значения атрибута А.

Зависимость, при которой каждый не ключевой атрибут зависит от всего составного ключа и не зависит от его частей, называется полной функциональной зависимостью. Если атрибут А зависит от атрибута В, а атрибут В зависит от атрибута С, но обратная зависимость при этом отсутствует, то зависимость С от А называется транзитивной.

Общее понятие нормализации подразделяется на несколько нормальных форм.

Информационный объект (сущность) находится в первой нормальной форме (НФ), когда все его атрибуты имеют единственное значение. Если в каком-либо атрибуте есть повторяющиеся значения, объект (сущность) не находится в 1НФ, и упущен, по крайней мере, еще один информационный объект (еще одна сущность).

Например, задано следующее отношение:

ПРЕДМЕТ (Код предмета, Название, Цикл, Объем часов, Преподаватели).

Это отношение не находится в 1НФ, так как атрибут ПРЕПОДАВАТЕЛИ подразумевает возможность наличия нескольких фамилий преподавателей в записи, относящейся к какому-то конкретному предмету, что соответствует участию нескольких преподавателей в ведении одной дисциплины.

Переведем атрибут с повторяющимися значениями в новую сущность, назначим ей первичный ключ (Код преподавателя) и свяжем с исходной сущностью ссылкой на ее первичный ключ (Код предмета). В результате получим две сущности, причем во вторую сущность добавятся характеризующие ее атрибуты:

ПРЕДМЕТ (Код предмета, Название, Цикл, Объем часов);

ПРЕПОДАВАТЕЛЬ (Код преподавателя, ФИО, Должность, Оклад, Адрес, Код предмета).

Полученные выражения соответствуют случаю, когда несколько преподавателей могут вести один предмет, но каждый преподаватель не может вести более одной дисциплины. А если учесть, что на самом деле один лектор может читать более одной дисциплины, так же как одну и ту же дисциплину могут читать несколько лекторов, необходимо отказаться от жесткой привязки преподавателя к предмету в сущности ПРЕПОДАВАТЕЛЬ, создав дополнительную сущность ИЗУЧЕНИЕ, которая будет показывать, как связаны между собой преподаватели и предметы:

ПРЕДМЕТ (Код предмета, Название, Цикл, Объем часов);

ПРЕПОДАВАТЕЛЬ (Код преподавателя, ФИО, Должность, Оклад, Адрес);

ИЗУЧЕНИЕ (Код предмета, Код преподавателя).

Информационный объект находится во второй нормальной форме (НФ), если он уже находится в первой нормальной форме и каждый его не идентифицирующий (описательный) атрибут зависит от всего уникального идентификатора информационного объекта. Если некий атрибут не зависит полностью от уникально го идентификатора информационного объекта, значит, он внесен в состав этого информационного объекта ошибочно и его не обходимо удалить. Нормализация в этом случае производится путем нахождения существующего информационного объекта, к которому данный атрибут относится, или созданием нового информационного объекта, в который атрибут должен быть помещен.

Возвратившись к последнему примеру, заметим, что атрибут Цикл, в сущности ПРЕДМЕТ, характеризующий принадлежность предмета к циклу гуманитарных, естественно-научных, общепрофессиональных или специальных дисциплин, не полностью зависит от уникального идентификатора Код предмета, так как разные предметы могут иметь одно и то же значение атрибута Цикл. Перенесем этот атрибут в новую сущность ЦИКЛ и получим четыре взаимосвязанных сущности:

ПРЕДМЕТ (Код предмета, Название, Объем часов, Код цикла);

ЦИКЛ (Код цикла, Название цикла);

ПРЕПОДАВАТЕЛЬ (Код преподавателя, ФИО, Должность, Оклад, Адрес);

ИЗУЧЕНИЕ (Код предмета, Код преподавателя).

Информационный объект находится в третьей нормальной форме (ЗНФ), если он уже находится во второй нормальной форме и ни один его описательный атрибут не зависит от каких-либо других описательных атрибутов. Атрибуты, зависящие от других не идентифицирующих атрибутов, нормализуются путем перемещения зависимого атрибута и атрибута, от которого он зависит, в новый информационный объект.

В данном случае не ключевые атрибуты Должность и Оклад находятся в транзитивной зависимости. Опасность такой зависимости состоит в том, что несколько человек могут работать в одной и той же должности. При изменении должностного оклада в этом

случае нужно будет менять данные в каждой записи, содержащей эту должность, следовательно, требуется создать новую сущность ДОЛЖНОСТЬ с находящимися в транзитивной зависимости атрибутами Название должности и Оклад и сделать ссылку от сущности ПРЕПОДАВАТЕЛЬ на сущность ДОЛЖНОСТЬ:

ПРЕДМЕТ (Код предмета, Название, Объем часов, Код цикла);

ЦИКЛ (Код цикла, Название цикла);

ПРЕПОДАВАТЕЛЬ (Код преподавателя, ФИО, Код должности, Адрес);

ДОЛЖНОСТЬ (Код должности, Название должности, Оклад);

ИЗУЧЕНИЕ (Код предмета, Код преподавателя).

**СРЕДСТВА ДОСТУПА К ДАННЫМ**

Чтобы пользователь чувствовал себя комфортно, время ожидания ответа на запрос к БД не должно превышать нескольких секунд. В связи с этим требованием специально разрабатываются методы ускорения выборки, позволяющие обойтись без полного перебора строк при выполнении реляционных операций модификации отношений и отбора данных.

Наиболее эффективны методы индексирования и хеширования значений ключей отношения.

*Индексирование* — логическая сортировка строк таблицы — заключается в создании вспомогательных файлов, содержащих упорядоченные списки значений ключей отношения со ссылками на строку отношения, в которой они находятся. Индексные файлы занимают дополнительную память, но резко ускоряют поиск благодаря применению метода половинного деления. Для одного от ношения может быть создано несколько индексов. Кроме того, можно создать индекс для нескольких отношений, если они содержат одинаковые атрибуты, что позволит ускорить выполнение операций соединения этих отношений.

Индексы позволяют находить строки, в которых значения ключевых полей совпадают с заданным значением или принадлежат заданному интервалу.

*Хеширование*— использование хэш-функций, которые вычисляют вес строки таблицы по значению ее ключевых атрибутов. Результат вычисления хэш-функции — целое число в диапазоне физических номеров строк таблицы.

Для организации доступа к данным при хешировании создается таблица с пустыми строками, которая заполняется следующим образом:

• по первичному ключу новой строки вычисляется значение хэш-функции и результат трактуется как номер строки в созданной таблице;

• если строка уже занята, производится проверка следующих строк по специальному алгоритму до тех пор, пока не будет обнаружено свободное место.

Аналогично производится поиск нужной строки:

• если после вычисления на месте в таблице, которое соответствует вычисленному значению, оказывается пустая строка, значит, искомой строки просто нет;

• если значение ключа совпало с искомым, поиск заканчивается;

• если же значение ключа не совпало с искомым, проверяются следующие строки таблицы до обнаружения строки с нужным ключом (в этом случае искомая строка найдена) или пустой строки (в этом случае искомая строка отсутствует).

Если таблица заполнена не более чем на 60 %, то для размещения новой или поиска существующей строки необходимо проверить в среднем не более двух ячеек. Хеширование используют для поиска строк по точному совпадению значения ключевого атрибута кортежа с нужным значением ключа.

**Контрольные вопросы**

1. Что такое нормализация таблиц, для какой цели проводят процедуру нормализации?
2. Какие формы нормализации вы знаете
3. Что такое индексирование? Достоинства, недостатки
4. Что такое хеширование? Достоинства, недостатки

**ТЕМА 1.3. ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ. СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ**

На этапе проектирования базы данных разработчик должен определить, из каких таблиц должна состоять база данных, какие данные нужно поместить в каждую таблицу и как связать таблицы.

Следовательно, в результате проектирования определяются логическая структура базы данных, т. е. состав реляционных таблиц, их структура и межтабличные связи.

Для создания базы данных необходимо располагать описанием выбранной предметной области, охватывающим реальные объекты и процессы, а также определить все необходимые источники информации для удовлетворения предполагаемых запросов пользователей и потребности в обработке данных. На основе такого описания определяются состав и структура данных предметной области, которые должны находиться в базе и обеспечивать выполнение необходимых запросов и задач пользователей. Структура данных предметной области может отображаться информационно логической моделью, на основе которой легко создается реляционная база данных.

Этапы проектирования и создания базы данных:

• построение информационно-логической модели данных предметной области;

• определение логической структуры реляционной базы данных;

• конструирование таблиц базы данных;

• создание схемы данных;

• ввод данных в таблицы (создание записей);

• разработка необходимых форм, запросов, макросов, моду лей, отчетов;

• разработка пользовательского интерфейса.

В процессе разработки модели данных необходимо выделить информационные объекты, соответствующие требованиям нормализации данных, и определить связи между ними. Полученная модель позволит создать реляционную базу данных без дублирования, в которой обеспечиваются однократный ввод данных при первоначальной загрузке и корректировках, а также целостность данных при внесении изменений.

При разработке модели данных используются два подхода.

**1.** Сначала определяются основные задачи, для решения которых строится база, выявляются потребности задач в данных и со ответственно определяются состав и структура информационных объектов.

**2.** Сразу устанавливаются типовые объекты предметной области. Наиболее рационально сочетание этих подходов, так как на начальном этапе, как правило, нет исчерпывающих сведений обо всех задачах. Использование такой технологии тем более оправдано, что гибкие средства создания реляционной базы данных допускают на любом этапе разработки внесение изменений и модифицирование ее структуры без ущерба для введенных ранее данных.

Процесс выделения информационных объектов предметной области, отвечающих требованиям нормализации, может производиться на основе интуитивного или формального подхода. Теоретические основы формального подхода разработаны известным американским ученым Дж. Мартином и изложены в его монографиях по организации баз данных.

При интуитивном подходе легко выявить информационные объекты, соответствующие реальным объектам, однако получаемая при этом информационно-логическая модель, как правило, требует дальнейших преобразований, в частности преобразования многозначных связей между объектами. При таком подходе в случае отсутствия достаточного опыта возможны существенные ошибки. Последующая проверка выполнения требований нормализации обычно показывает необходимость уточнения информационных объектов.

Рассмотрим формальные правила выделения информационных объектов:

на основе описания предметной области выявить документы и их атрибуты, подлежащие хранению в базе данных;

определить функциональные зависимости между атрибутами; выбрать все зависимые атрибуты и указать для каждого все его ключевые атрибуты, т. е. атрибуты, от которых он зависит;

сгруппировать атрибуты, одинаково зависимые от ключевых атрибутов. Полученные группы зависимых атрибутов вместе с их ключевыми атрибутами образуют информационные объекты.

При определении логической структуры реляционной базы данных на основе модели каждый информационный объект адекватно отображается реляционной таблицей, а связи между этими таблицами соответствуют связям между информационными объектами.

В процессе создания БД сначала конструируются таблицы, соответствующие информационным объектам построенной модели данных. Далее может создаваться схема данных, в которой фиксируются существующие логические связи между таблицами, соответствующие связям информационных объектов. В схеме данных могут быть заданы параметры поддержания целостности базы данных, если модель была разработана в соответствии с требования ми нормализации. Целостность данных означает, что в БД установлены и корректно поддерживаются взаимосвязи между записями разных таблиц при загрузке, добавлении и удалении записей в связанных таблицах, а также при изменении значений ключевых полей.

После формирования схемы данных осуществляется ввод непротиворечивых данных из документов предметной области.

На основе созданной базы формируются необходимые запросы, формы, макросы, модули, отчеты, производящие требуемую обработку данных и их представление.

С помощью встроенных средств и инструментов базы данных создается пользовательский интерфейс, позволяющий управлять процессами ввода, хранения, обработки, обновления и представления информации.

**Проектирование базы данных на основе модели типа объект- отношение**

Имеется целый ряд методик создания информационно-логических моделей. Наиболее популярна в настоящее время методика с использованием ERD (диаграммы, которые в русскоязычной литературе называют объект — отношение, либо сущность—связь) . Модель с использованием ERD была предложена Членом в 1976 г. К настоящему времени разработано несколько ее разновидностей, но все они базируются на графических диаграммах, предложенных Членом, которые конструируются из небольшого числа компонентов и благодаря наглядности представления широко используются.

Рассмотрим используемые при проектировании терминологию и обозначения.

Сущность — реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению.

Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа, т. е. каждая сущность должна:

• иметь уникальное имя, причем это имя должно всегда однозначно интерпретироваться (определять сущность), и наоборот, одна интерпретация не может применяться к различным именам, если только они не являются псевдонимами;

• обладать одним или несколькими атрибутами, которые либо принадлежат ей, либо наследуются ею через связь;

• обладать одним или несколькими атрибутами, которые однозначно идентифицируют каждый ее экземпляр.

Сущность может быть зависимой и независимой.

<ИМЯ СУЩНОСТИ>

<ИМЯ СУЩНОСТИ>

Признаком зависимой сущности служит наличие у нее наследуемых через связь атрибутов. Каждая сущность может обладать любым количеством связей с другими сущностями модели.

Связь — поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. При этом одна из участвующих в связи сущностей — независимая — называется родительской, а другая — зависимая — называется дочерней, или сущностью-потомком. Как правило, каждый экземпляр родительской сущности ассоциирован с произвольным (в том числе нулевым) числом экземпляров дочерней сущности. Каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя.

Связи дается имя, выражаемое глаголом и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Каждая связь имеет определение. Определение связи образуют соединением имени сущности-родителя, имени связи, выражения степени связи и имени сущности-потомка.

Например, связь продавца с контрактом может быть определена следующим образом:

• продавец может получить вознаграждение за один или более контрактов;

• контракт должен быть инициирован одним продавцом.

На диаграммах связь изображается отрезками. Концы этих отрезков с помощью специальных обозначений указывают степень связи. Кроме того, характер линии (штриховая или сплошная) указывает обязательность связи.

Атрибут — любая характеристика сущности, значимая для рассматриваемой предметной области. Он предназначен для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет собой тип характеристик (свойств), ассоциированных с множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, пар предметов и т.д.). Экземпляр атрибута — это определенная характеристика конкретного экземпляра сущности. Экземпляр атрибута определяется типом характеристики (например, цвет) и ее значением (например, лиловый), называемым значением атрибута. В ЕRD-модели атрибуты ассоциируются с конкретными сущностями. Каждый экземпляр сущности должен обладать одним конкретным значением для каждого своего атрибута.

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

Уникальный идентификатор — это атрибут или совокупность атрибутов и/или связей, однозначно характеризующая каждый экземпляр данного типа сущности. В случае полной идентификации экземпляр данного типа сущности полностью идентифицируется своими собственными ключевыми атрибутами, в противном случае в идентификации участвуют также атрибуты сущности-родителя.

Каждый атрибут идентифицируется уникальным именем, выражаемым существительным, описывающим представляемую атрибутом характеристику. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Атрибуты, определяющие первичный ключ, размещаются вверху списка.

Каждая сущность должна обладать хотя бы одним возможным ключом. Возможный ключ сущности — это один или несколько атрибутов со значениями, однозначно определяющими каждый ее экземпляр. При существовании нескольких возможных ключей один из них обозначается в качестве первичного, а остальные — как альтернативные.

Большинство связей относятся к категории бинарных и имеют место между двумя сущностями.

Среди бинарных связей существуют три фундаментальных вида связи: один-к-одному (1:1), один-ко-многим (1:M), многие-ко-многим (M:M). Связь один-к-одному (1:1) существует, когда один экземпляр одной сущности связан с единственным экземпляром другой сущности. Связь один-ко-многим (1:M) имеет место, когда один экземпляр одной сущности связан с одним или более экземпляром другой сущности и каждый экземпляр второй сущности связан только с одним экземпляром первой сущности. Связь многие-ко-многим (М:N) существует, когда один экземпляр одной сущности связан с одним или более экземпляром другой сущности и каждый экземпляр второй сущности связан с одним или более экземпляром первой сущности.

В условных связях в отличие от безусловных могут существовать экземпляры сущности, которые в связи не принимают участия. Если связь условная с обеих сторон, она называется биусловной.

Все связи требуют описания. Описание должно обеспечивать:

* идентификатор связи;
* формулировку имен связи с точки зрения каждой участвующей сущности;
* вид связи (множественность и условность);
* формулировку того, как связь была формализована.

Цель формализации связи состоит в том, чтобы позволить установить связь экземпляра одной сущности с экземпляром другого. Формализация связи выполняется размещением вспомогательных атрибутов в соответствующих сущностях модели.

Все сущности относятся к одному из четырех классов:

* стержневые;
* ассоциативные;
* характеристические;
* обозначающие.

*Стержневая сущность* (стержень) представляет собой независимую сущность.

*Ассоциативная сущность* (ассоциация) - это сущность, формализующая связь вида M:N между двумя или более сущностями или связь вида 1:1 между экземплярами сущностей.

*Характеристическая сущность* (характеристика) представляет собой сущность, формализующую связь вида 1:M или 1:1. Единственная цель характеристики в рамках рассматриваемой предметной области состоит в описании или уточнении некоторой другой сущности.

*Обозначающая сущность* (обозначение) - это сущность, также формализующая связь вида 1:M или 1:1 между двумя сущностями, но отличающаяся от характеристики тем, что не зависит от обозначаемой сущности.

К числу более сложных элементов ER-модели относятся подтипы и супертипы сущностей. Сущность может быть расщеплена на два или более взаимно исключающих подтипа, каждый из которых имеет общие атрибуты и/или связи. Эти общие атрибуты и/или связи явно определяются один раз на более высоком уровне. В подтипах могут определяться собственные атрибуты и/или связи. Сущность, на основе которой определяются подтипы, называется супертипом. Подтипы должны образовывать полное множество, т.е. любой экзем-пляр супертипа должен относиться к некоторому подтипу. Аналогично языкам объектно-ориентированного программирования вводится возможность наследова-ния типа сущности исходя из одного или нескольких супертипов.

В случае очень большого числа сущностей и связей между ними применяется менее наглядный, чем язык ER-диаграмм, но более содержательный язык инфологического моделирования, в котором сущности и связи представляются предложениями вида

*СУЩНОСТЬ (атрибут 1, атрибут 2 , ..., атрибут n)*

*СВЯЗЬ [СУЩНОСТЬ S1, СУЩНОСТЬ S2, ...] (атрибут 1,..., атрибут n).*

**Контрольные вопросы**

1. Каковы задачи, решаемые на этапе инфологического проектирования?
2. В чем состоит отличие понятия типа сущности и элемента сущности?
3. Каковы способы представления сущности?
4. Каковы правила атрибутов?
5. Как классифицируются атрибуты?
6. Что такое безусловная, условная, биусловная, рекурсивная связь?
7. Каковы фундаментальные виды связей?
8. Как формализуется связь 1:1?
9. Как формализуется связь 1:M?
10. Как формализуется связь M:N?
11. Что такое подтип и супертип?
12. Что такое композиция связей?

**Задания**

1. Выделить необходимый набор сущностей, отражающих предметную область и информационные потребности пользователей.

2. Определить необходимый набор атрибутов каждой сущности, выделив идентифицирующие атрибуты.

3. Классифицировать атрибуты каждой сущности (описательные, указывающие, вспомогательные).

4. Определить сущности вида подтип/супертип, где это необходимо.

5. Определить связи между сущностями.

6. Проанализировав структуру связей, исключить избыточные.

7. Определить множественность и условность связей.

8. Дать формулировку связей с точки зрения каждой участвующей сущности.

9. Формализовать связи вида 1:1, 1:M, M:N.

10. Классифицировать сущности, разделив их на стержневые, ассоциативные, характеристические, обозначающие.

11. Построить ER-диаграмму модели базы данных.

12. Описать модель базы данных на языке инфологического проектирования.

**РАЗДЕЛ 2. ИСПОЛЬЗОВАНИЕ СУБД АССЕSS ДЛЯ СОЗДАНИЯ БАЗ ДАННЫХ**

**ТЕМА 2.1. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ И ВОЗМОЖНОСТИ СУБД АССЕSS**

Группа реляционных СУБД представлена на рынке программных продуктов очень широко. Это, например, такие системы, как Рагаdох, Сlагiоп, dВАSE, FохВАSE, FохРго, Сliррег, Ассеss. Важнейшей характеристикой любой СУБД является используемый в ней тип транслятора (интерпретатор или компилятор). Программы, написанные для системы—интерпретатора, не работают без наличия самой этой системы. В настоящее время скорость работы таких программ не уступает скорости программ, сгенерированных компилятором. Бесспорным преимуществом интерпретаторов для программистов является удобство разработки и отладки программных продуктов, а также освоение языка. Из перечисленных СУБД dВАSE FохРго, Ассеss являются интерпретаторами, а Сlipper — компилятором. В пакетах dВАSE и FохРго имеется компилятор, позволяющий при желании сформировать ЕХЕ-файлы готовых программ. Недостатком систем-компиляторов являются большие суммарные затраты времени на многократную компиляцию и сборку (линковку) исходных модулей программы при ее отладке.

СУБД Ассеss (фирма Мiсгоsoft) имеет достаточно высокие скоростные характеристики и входит в состав чрезвычайно популярного в нашей стране и за рубежом пакета Мiсгоsoft Offiсе. Набор команд и функций, предлагаемых разработчикам программных продуктов в среде Ассеss, по мощи и гибкости отвечает большинству современных требований к представлению и обработке данных. В Ассеss поддерживаются разнообразные всплывающие и многоуровневые меню, работа с окнами и мышью, реализованы функции низкоуровневого доступа к файлам, управления цветами, настройки принтера, представления данных в виде электронных таблиц и т. п. Система также обладает средствами быстрой генерации экранов, отчетов и меню, поддерживает язык управления запросами SQL, имеет встроенный язык Visual Basik for Applications (VВА), хорошо работает в сети. СУБД Ассеss позволяет использовать другие компоненты пакета Мiсгоsoft Оffiсе, такие как текстовый процессор Word for Windows, электронные таблицы Ехсеl и т.д.

Перечисленные факторы определили выбор СУБД Ассеss в качестве среды для практического изучения вопросов проектирования баз данных.

Приведем некоторые из средств Мiсгоsoft Ассеss, существенно упрощающие разработку приложений.

1. **Процедуры обработки событий и модули форм и отчетов**. На встроенном языке VВА можно писать процедуры обработки событий, возникающих в формах и отчетах. Процедуры обработки событий хранятся в модулях, связанных с конкретными формами и отчетами, в результате чего код становится частью макета формы или отчета. Кроме того, существует возможность вызова функции VВА свойством события.

2. **Свойства, определяемые в процессе выполнения**. С помощью макроса или процедуры обработки событий можно определить практически любое свойство формы или отчета в процессе выполнения в ответ на возникновение события в форме или отчете.

З. **Модель событий**. Модель событий, похожая на используемую в языке Мiсгоsoft VisuaI Ваsic позволяет приложениям реагировать на возникновение различных событий, например нажатие клавиши на клавиатуре, перемещение мыши или истечение определенного интервала времени.

4**. Использование обработки данных с помощью VBA**. С помощью языка VBA можно определять и обрабатывать различные объекты, в том числе, таблицы, запросы, поля, индексы, связи, формы, отчеты и элементы управления.

5. **Построитель меню**. Предназначен для помощи при создании специальных меню в приложениях. Кроме того, специальные меню могут содержать подменю.

б. **Улучшенные средства отладки**. Помимо установки точек прерывания и пошагового выполнения программ на языке VВА, можно вывести на экран список всех активных процедур. Для этого следует выбрать команду ***Вызовы*** в меню ***Вид*** или нажать кнопку (***Вызовы)*** на панели инструментов.

7. **Процедура обработки ошибок**. Помимо традиционных способов обработки ошибок возможно использование процедуры обработки события Еrrorдля перехвата ошибок при выполнении программ и макросов.

8. **Улучшенный интерфейс защиты**. Команды и окна диалога защиты упрощают процедуру защиты и смены владельца объекта.

9. **Программная поддержка механизма** **OLE.** С помощью механизма OLE можно обрабатывать объекты из других приложений.

10. **Программы-надстройки**. С помощью VВА можно создавать программы-надстройки, например нестандартные мастера и построители. Мастер — средство Мiсгоsoft Ассеss, которое сначала задает пользователю вопросы, а затем создает объект (таблицу, запрос, форму, отчет и т.д.) в соответствии с его указаниями.

Диспетчер надстроек существенно упрощает процедуру установки программ-надстроек в Мiсгоsoft Ассеss

**Мастера Ассеss**

Ассеss позволяет даже мало подготовленному пользователю создать свою БД, обрабатывать данные с помощью форм, запросов и отчетов, проводить анализ таблиц БД и выполнять ряд дру- гих работ. Практически для любых работ с БД в Ассеss имеется свой мастер, который помогает их выполнять.

***Мастер по анализу таблиц*** позволяет повысить эффективность базы данных за счет нормализации данных. Он разделяет ненормализованную таблицу на две или несколько таблиц меньшего размера, в которых данные сохраняются без повторения.

***Мастера по созданию форм и отчетов*** упрощают и ускоряют процесс создания многотабличных форм и отчетов. Новые форма и отчет могут наследовать примененный к таблице-источнику записей фильтр.. Мастера по разработке форм и отчетов автоматически создают инструкцию SQL, определяющую источник записей для формы или отчета, поэтому отпадает необходимость в создании запроса

Для изменения вида формы, отчета или отдельных элементов может быть использован мастер, вызываемый кнопкой (Автоформат).

***Мастер подстановок*** создает в поле таблицы раскрывающийся список значений из другой таблицы для выбора и ввода нужного значения. для создания такого поля со списком достаточно в режиме конструктора таблицы выбрать тип данных этого поля — *Мастер подстановок*. Мастер подстановок можно вызвать в режиме таблицы командой меню *Вставка/ Столбец подстановок*. Созданный в данном поле таблицы список наследуется при включении этого поля в форму.

***Мастера по импорту/экспорту*** позволяют просматривать данные при импорте/экспорте текста или электронных таблиц, а также при экспорте данных Мiсгоsoft Ассеss в текстовые файлы.

***Мастер защиты*** при необходимости эвакуирует данные, для чего создает новую базу данных, копирует в нее все объекты из исходной базы данных, снимает все права, присвоенные членам группы пользователей, и шифрует новую базу данных. После завершения работы мастера администратор может присвоить новые права доступа пользователям и группам пользователей.

***Мастер по разделению базы данных*** позволяет разделить ее на два файла, в первый из которых помещаются таблицы, а во второй запросы, формы, отчеты, макросы и модули. При этом пользователи, работающие в сети, имея общий источник данных, смогут устраивать формы, отчеты и другие объекты, используемые для обработки данных, по своему усмотрению.

**Основные компоненты СУБД Ассеss**

Основными компонентами (объектами) базы данных являются таблицы, запросы, формы, отчеты, макросы и модули.

**Таблица** — фундаментальная структура системы управления реляционными базами данных. В Мiсгоsoft Ассеss таблица — это объект, предназначенный для хранения данных в виде записей (строк) и полей (столбцов). При этом каждое поле содержит от дельную часть записи (например, фамилию, должность или инвентарный номер). Обычно каждая таблица используется для хранения сведений по одному конкретному вопросу (например, о сотрудниках или заказах).

**Запрос** — вопрос о данных, хранящихся в таблицах, или инструкция на отбор записей, подлежащих изменению.

Перечислим типы запросов, которые могут быть созданы с по мощью Мiсгоsoft Ассеss

*запрос-выборка*, задающий вопрос о данных, хранящихся в таблицах, и представляющий полученный динамический набор в режиме формы или таблицы без изменения данных. Изменения, внесенные в динамический набор, отражаются в базовых таблицах;

*запрос-изменение*, изменяющий или перемещающий данные. К этому типу относятся запрос на добавление записей, запрос на удаление записей, запрос на создание таблицы или запрос на ее обновление;

*перекрестные запросы*, предназначенные для группирования данных и представления их в компактном виде;

*запрос с параметрами*, позволяющий определить одно или несколько условий отбора во время выполнения запроса;

*запросы SQL* которые могут быть созданы только с помощью инструкций SQL в режиме SQL: запрос-объединение, запрос к серверу и управляющий запрос. Язык SQL- это язык запросов, который часто используется при анализе, обновлении и обработке реляционных баз данных.

**Форма** - это объект, в котором можно разместить элементы управления, предназначенные для ввода, изображения и изменения данных в полях таблиц.

**Отчет** - это объект, который позволяет представлять определенную пользователем информацию в определен ном виде, просматривать и распечатывать ее.

**Макрос** - одна или несколько макрокоманд, которые можно использовать для автоматизации конкретной задачи.

**Макрокоманда**- основной строительный блок макроса; само­стоятельная инструкция, которая может быть объединена с други­ми макрокомандами для автоматизации выполнения задачи.

**Модуль -** набор описаний, инструкций и процедур, сохранен­ных под одним именем. В Мiсrosоft Access имеется три типа моду­лей: формы, отчета и общий.

**Типы данных СУБД Access**

Для каждого поля таблиц базы данных необходимо указывать тип данных. Тип данных определяет вид и диапазон допустимых значений, которые могут быть введены в поле, а также объем памяти, выделяющийся для этого поля. Перечень типов данных полей и описание значений, сохраняемых в таких полях.

|  |  |
| --- | --- |
| **Тип данных** | **Описание** |
| Текстовый (Значение по умолчанию) | Текст или числа, не требующие проведения расчетов, например номера телефонов (до 255 знаков) |
| Числовой | Числовые данные различных форматов, используемые для проведения расчетов |
| Дата/время | Для хранения информации о дате и времени с 100 по 9999 год включительно |
| Денежный | Денежные значения и числовые данные, используемые в математических расчетах, проводящихся с точностью до 15 знаков в целой и до 4 знаков в дробной части |
| Поле MEMO | Для хранения комментариев; до 65535 символов |
| Счетчик | Специальное числовое поле, в котором Access автоматически присваивает уникальный порядковый номер каждой записи. Значения полей типа счетчика обновлять нельзя |
| Логический | Может иметь только одно из двух возможных значений (TrueFalse, Да/Нет) |
| Поле объекта OLE | Объект (например, электронная таблица Microsoft Excel, документ Microsoft Word, рисунок, звукозапись или другие данные в двоичном формате), связанный или внедренный в таблицу Access |
| Гиперссылка | Строка, состоящая из букв и цифр и представляющая адрес гиперссылки. Адрес гиперссылки может состоять максимум из трех частей: текст, выводимый в поле или в элементе управления; путь к файлу (в формате пути UNC) или к странице (адрес URL). Чтобы вставить адрес гиперссылки в поле или в элемент управления, выполните команду Вставка, Гиперссылка |
| Мастер подстановок | Создает поле, в котором предлагается выбор значений из списка или из поля со списком, содержащего набор постоянных значений или значений из другой таблицы. Это в действительности не тип поля, а способ хранения поля |

**Примечание**: Microsoft Access обеспечивает возможность хранения в полях таблиц специального значения Null, называемого пустым значением, для обозначения некоторого неизвестного значения. При этом очень важно различать строки со значением Null и пустые строки (“”). Пустое значение имеет особые свойства. Оно не может быть равно никакому другому значению, в том числе и другому пустому значению, т.е. нельзя связать две таблицы по пустому значению. Кроме того, значения Null не учитываются в групповых операциях.

**Контрольные вопросы**

1. Перечислите основные характеристики и возможности
2. Какие компоненты входят в состав СУБД Access?
3. Какие виды мастеров, входящих в, вы знаете?
4. Какие типы данных используются при работе с СУБД Access?

**ТЕМА 2.2. СОЗДАНИЕ ТАБЛИЦ В СУБД ACCESS**

Таблицы создаются пользователем для хранения данных. Каж­дому объекту концептуальной модели предметной области соот­ветствует одна таблица, которая состоит из полей (столбцов) и записей (строк). Каждое поле содержит одну характеристику (один атрибут) объекта предметной области. В записи собраны сведения об одном экземпляре этого объекта.

Работа по созданию базы данных на персональном компьюте­ре (ПК) начинается с создания таблиц. После нажатия кнопки [Создать] в окне *База данных* разработчику предоставляется воз­можность выбора одного из ПЯТИ способов создания таблицы.

Если для создания таблицы выбран режим конструктора, то появляется окно *Таблица]: таблица,* в котором определяется струк­тура создаваемой таблицы базы данных.

Для определения поля в открывшемся окне задаются *Имя поля, Тип данных, Описание* (в виде краткого комментария), а также в

разделе *Свойства поля* задаются общие свойства - на закладке *Общие* и тип элемента управления - на закладке *Подстановка.*

Каждое поле в таблице должно иметь уникальное имя, удов­летворяющее соглашениям об именах объектов в Access и являю­щееся комбинацией из букв, цифр, пробелов и специальных симво­лов. Максимальная длина имени - 64 Символа.

Тип данных определяется значениями, которые предполагает­ся вводить в поле, и операциями, которые будут выполняться с этими значениями. В Access допускается использование восьми типов данных. Список возможных типов данных каждого поля вы­зывается нажатием соответствующей кнопки.

Общие свойства поля задаются на закладке *Общие* для каждого поля и зависят от выбранного типа данных.

Наиболее важные свойства полей:

*Размер поля* - определяет максимальный размер данных, со­храняемых в поле. Рекомендуется задавать минимально допусти­мый размер поля, так как сохранение таких полей требует мень­ше памяти и обработка выполняется быстрее;

*Формат поля* - является форматом отображения заданного типа данных и задает правила представления этих данных при выводе их на экран или печать. Конкретный формат выбирается в рас­крывающемся списке значений свойства *Формат поля.* Для число­вого и денежного типов данных задается число знаков после запя­той (от 0до 15);

*Подпись поля* - задает текст, который выводится в таблицах, формах, отчетах;

*Условие на значение* - позволяет осуществлять контроль ввода данных, задает ограничения на вводимые значения, при наруше­нии условий запрещает ввод и выводит текст, заданный свой­ством *Сообщение об ошибке;*

*Сообщение об ошибке* - задает текст сообщения, выводимый на экран при нарушении ограничений, заданных свойством *Условие на значение.*

Тип элемента управления - это свойство, которое задается на закладке *Подстановка* в окне конструктора таблиц и определяет, будет ли отображаться поле в таблице и в какой форме (в виде поля, списка или поля со списком). Таким образом определяется тип элемента управления, используемого по умолчанию для отображения поля. Если для отображения поля выбран тип элемента управления *Список* или *Поле со списком,* то на закладке *Подстанов­ка* появляются дополнительные свойства, которые определяют ис­точник данных для строк списка и ряд других его характеристик.

Если при определении типа поля был выбран мастер подста­новок, то им и будут заполнены значения свойств на закладке *Подстановка.*

Определение первичного ключа

Уникальный (первичный) ключ таблицы может быть простым или составным, включающим в себя несколько полей. Для опре­деления ключа выделяются поля, составляющие его, и на панели инструментов нажимается кнопка [Ключевое поле] ИЛИ выполня­ется команда *Правка Ключевое поле.*

Если для установки ключевого поля используется мастер таб­лиц, то необходимо задать тип используемых в ключевом поле данных.

Для ключевого поля автоматически строится индекс. Убедиться в этом можно, просмотрев информацию об индексах таблицы.

Окно *Индексы* вызывается щелчком мыши на кнопке [Индек­сы] просмотра и редактирования индексов или выполнением ко­манды *Вид Индексы.* В этом окне индекс первичного ключа имеет имя *Primary Кеу.* В столбце *Имя поля* этого окна перечисляются имена полей, составляющие индекс. Индекс ключевого поля всегда уникален и не допускает пус­тых полей в записях.

**Задание**

1. Создайте таблицу следующего вида**:**

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| № | Счетчик |  |
| Персонаж | Текстовый |  |
| Профессия | Текстовый |  |
| Особые приметы | Текстовый |  |
| Герой | Логический | Положительный или отрицательный герой |

1. Заполните таблицу по образцу**:**

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| № | Счетчик |  |
| Персонаж | Текстовый |  |
| Профессия | Текстовый |  |
| Особые приметы | Текстовый |  |
| Герой | Логический | Положительный или отрицательный герой |

1. Введите в таблицу следующие данные:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Персонаж** | **Профессия** | **особые приметы** | **герой** |
| 1 | Буратино | деревянный человечек | длинный нос | Да |
| 2 | Папа Карло | Шарманщик |  | Да |
| 3 | Карабас Барабас | директор кукольного театра | длинная борода, достающая до пола | Нет |
| 4 | Лиса Алиса | Мошенница | хромая на одну ногу | Нет |
| 5 | Кот Базилио | Мошенник | слепой на оба глаза | Нет |
| 6 | Мальвина | артистка театра | девочка с голубыми волосами | Да |
| 7 | Дуремар | Фармацевт | характерный запах тины | Нет |
| 8 | Тортилла | хранительница золотого ключика | черепаха | Да |

**Схема данных в Access**

Структура реляционной базы данных в Access задается схемой данных, которая имеет иерархическую структуру и называется канонической реляционной моделью предметной области.

Схема данных графически отображается в отдельном окне, в котором таблицы представлены списками полей, а связи - лини­ями между полями разных таблиц.

При построении схемы данных Access автоматически определя­ет по выбранному полю тип связи между таблицами. Если поле, по которому нужно установить связь, является уникальным ключом как в главной таблице, так и в подчиненной, Access устанавливает связь типа один к одному. Если поле связи является уникальным ключом в главной таблице, а в подчиненной таблице является не ключевым или входит в составной ключ, Access устанавливает связь типа один ко многим от главной таблицы к подчиненной.

Кроме указанных типов связей в Access существуют связи-объе­динения, обеспечивающие объединение записей таблиц не по ключевому полю, а в следующих случаях:

• связываемые записи в обеих таблицах совпадают (связи уста­навливаются по умолчанию);

• для всех записей первой таблицы, для которых отсутствуют связи со второй таблицей, устанавливаются связи с пустой запи­сью второй таблицы;

• для всех записей второй таблицы, для которых отсутствуют связи с первой таблицей, устанавливаются связи с пустой запи­сью первой таблицы.

**Обеспечение целостности данных**

При создании схемы данных пользователь включает в нее таб­лицы и устанавливает связи между ними. Причем для связей ти­пов ОДИН к одному и ОДИН КО многим можно задать параметр, обеспечивающий целостность данных, а также автоматическое кас­кадное обновление или удаление связанных записей.

Обеспечение целостности данных означает выполнение для взаимосвязанных таблиц следующих условий корректировки базы данных:

• в подчиненную таблицу не может быть добавлена запись, для которой не существует в главной таблице ключа связи;

• в главной таблице нельзя удалить запись, если не удалены связанные с ней записи в подчиненной таблице;

• изменение значений ключа связи главной таблицы должно приводить к изменению соответствующих значений в записях под­чиненной таблицы,

В случае если пользователь нарушил эти условия в операциях обновления или удаления данных в связанных таблицах, Access выводит соответствующее сообщение и не допускает выполнения операции. Access автоматически отслеживает целостность данных, если между таблицами в схеме данных установлена связь с пара­метрами обеспечения целостности. При вводе некорректных дан­ных в связанные таблицы выводится соответствующее сообщение. Access не позволяет создавать связи с параметрами обеспечения целостности в схеме данных, если ранее введенные в таблицы данные не отвечают требованиям целостности.

Отметим, что установление между двумя таблицами связи типа один-к-одному или один-ко-многим и задание параметров целост­ности данных возможно только при следующих условиях:

• связываемые поля имеют одинаковый тип данных, причем Имена полей могут быть различными;

1. обе таблицы сохраняются в одной базе данных Access;

• главная таблица связывается с подчиненной по первичному простому или составному ключу (уникальному индексу) главной таблицы.

Если для выбранной связи обеспечивается поддержание цело­стности, то можно задать режимы каскадного обновления и уда­ления связанных записей.

В режиме каскадного обновления связанных записей при изме­нении значения в поле связи главной таблицы Access автомати­чески изменит значения в соответствующем поле в подчиненных записях

В режиме каскадного удаления связанных записей при удале­нии записи из главной таблицы Access выполняет каскадное уда­ление подчиненных записей на всех уровнях.

**ТЕМА 2.3. ИНДЕКСИРОВАНИЕ: ПОНЯТИЕ ИНДЕКСА,**

**ТИПЫ ИНДЕКСНЫХ ФАЙЛОВ**

Основное назначение индексов состоит в обеспечении эффективного прямого доступа к записи таблицы по ключу. Различают индексированный файл и индексный файл. Индексированный файл - это основной файл, содержащий данные отношения, для которого создан индексный файл.

*Индексный файл* - это файл особого типа, в котором каждая запись состоит из двух значений: данных и указателя. Данные представляют поле, по которому производится индексирование, а указатель осуществляет связывание. соответствующим кортежем индексированного файла. Если индексирование осуществляется по ключевому полю, то индекс называется *первичным.* Такой индекс к тому же обладает свойством уникальности, т. е. не содержит дубликатов ключа.

Обычно индекс определяется для одного отношения, и ключом являете: значение простого или составного атрибута.

Основное преимущество использования индексов заключается в значительном ускорении процесса выборки данных, а основной недостаток - в замедлении процесса обновления данных. Действительно, при каждом добавлении новой записи в индексированный файл потребуется также добавит Новый индекс (а это дополнительное время) в индексный файл. Поэтом: При выборе поля индексирования всегда важно уточнить, который из двух показателей важнее: скорость выборки или скорость обновления.

Индексы позволяют:

1) осуществлять последовательный доступ к индексированному файлу в со­ответствии со значениями индексного поля для составления запросов на поиск наборов записей;

2) осуществлять прямой доступ к отдельным записям индексированного файла на основе заданного значения индексного поля для составления запросов для заданных значений индекса;

3) организовать запросы, не требующие обращения к индексированному файлу, а лишь приводящие к проверке наличия данного значения в ин­дексном файле.

Поскольку при выполнении многих операций языкового уровня требуется сортировка отношений в соответствии со значениями некоторых атрибутов, полезным свойством индекса является обеспечение последовательного Про­смотра кортежей отношения в диапазоне значений ключа в порядке возрас­тания или убывания значений ключа.

В зависимости от организации индексной и основной областей различают два типа файлов: индексно-прямые файлы и индексно-последовательные файлы. При рассмотрении определенной технологии будем, прежде всего, обращать внимание на ее достоинства и слабые места, а также на поддержи­ваемые ею способы поиска информации, вставки и удаления записи.

**Индексно-прямые файлы**

В индексно-прямых файлах основная область содержит последовательность записей одинаковой длины, расположенных в произвольном порядке, а ин­дексная запись содержит *значение первичного ключа* и *порядковый номер записи в основной области,* которая имеет данное значение первичного ключа.

Так как индексные файлы строятся для первичных ключей, однозначно оп­ределяющих запись, то в индексно-прямых файлах для каждой записи в ос­новной области существует только одна запись из индексной области. Такой индекс называется *плотным.* Все записи в индексной области упорядоченный по значению ключа, поэтому можно применить более эффективные спосо­бы поиска в упорядоченном пространстве.

Наиболее эффективным алгоритмом поиска на упорядоченном массиве яв­ляется бинарный поиск. При этом все пространство поиска разбивается пополам, и так как оно строго упорядочено, то сначала определяется, не является ли срединный элемент искомым, а если нет, то дается оценка в какой половине его надо искать. Далее установленная половина также де­лится пополам и производятся аналогичные действия, и так до тех пор, пока не будет обнаружен искомый элемент.

В данном случае двоичный алгоритм поиска при меняется к индексному файлу, а потом путем прямой адресации происходит обращение к основной области уже по конкретному номеру записи.

Операция добавления осуществляет запись в конец основной области. В индексной области при этом производится занесение информации так, чтобы не нарушать упорядоченности. Поэтому вся индексная область файла разбивается на блоки и при начальном заполнении в каждом блоке остается свободная область (процент расширения).

Блок, в который должен быть занесен индекс, копируется в оперативную память, где производится вставка новой записи, и измененный блок записывается обратно на диск.

Естественно, в процессе добавления новых записей процент расширения постоянно уменьшается. Когда исчезает свободная область, возникает переполнение индексной области. В этом случае возможны два решения: либо перестроить заново индексную область, либо организовать область переполнения для индексной области, в которой будут храниться не поместившиеся в основную область записи. Однако первый способ потребует дополнительного времени на перестройку индексной области, а второй увеличит время на доступ к произвольной записи и потребует организации дополнительных ссылок в блоках на область переполнения.

Именно поэтому при проектировании физической базы данных так важно заранее как можно точнее определить объемы хранимой информации, спрог­нозировать ее рост и предусмотреть соответствующее расширение области хранения.

При удалении записи возникает следующая последовательность действий: запись в основной области помечается как удаленная, в индексной области соответствующий индекс уничтожается физически, то есть записи индекс­ного файла, следующие за удаленной записью, перемещаются на ее место, и блок, в котором хранился данный индекс, заново записывается на диск. При этом количество обращений к диску для этой операции такое же, как и при добавлении новой записи.

**Индексно-последовательные файлы**

Если файлы поддерживаются в отсортированном состоянии с момента их создания, то для работы с ними может быть использован другой подход с технологией построения индексного файла, несколько отличной от выше­рассмотренной. Принципы внутреннего упорядочения и блочности по­строения таких файлов позволяют уменьшить количество хранимых индек­сов за счет того, что в индексном файле не содержатся. Указатели на все записи индексированного файла. Таким образом, в этом случае индекс по­лучается *неплотным* или *разреженным.*

Одним из преимуществ неплотных индексов является их малый размер по сравнению с плотными индексами, так как они содержат меньшее число записей. Это позволяет просматривать содержимое базы данных с большей скоростью.

Индексная запись для таких файлов должна содержать: значение ключа пер­вой записи блока и номер блока с этой записью.

Теперь по заданному значению первичного ключа в индексной области тре­буется отыскать уже нужный блок. Так как все записи упорядочены, то зна­чение первой записи блока позволяет быстро определить, в каком блоке на­ходится искомая запись. Все остальные действия происходят в основной области. При переходе к неплотному индексу время доступа уменьшается практически в полтора раза.

При таком подходе новая запись должна заноситься сразу в требуемый блок на требуемое место. Данное занесение осуществляется в оперативной памяти, куда считывается блок основной памяти, который вследствие упорядо­ченности записей по значению ключа должен принять эту запись. Содержи­мое считанного блока корректируется, и затем он снова записывается на диск на старое место. Естественно, что для добавления записей уже блок основной области должен иметь свободное место. При внесении новой записи индексная область не корректируется.

Уничтожение записи происходит путем ее физического удаления из основной области, при этом индексная область обычно не корректируется, даже если удаляется первая запись блока.

**ТЕМА 2.4. ОБРАБОТКА ДАННЫХ В БАЗЕ**

Microsoft Access позволяет создавать таблицы одним из двух способов: самостоятельно или при помощи мастера по разработке таблиц. В случае первого варианта создание новой таблицы происходит в режиме конструктора таблиц (в этом же режиме производится модернизация уже существующих таблиц).

Режим конструктора позволяет задать свойства строк таблицы (записей). Для каждого из полей записи можно ввести имя, тип данных и краткое описание. После того, как выбран тип поля можно задать свойства поля в соответствии с выбранным типом. Этими свойствами могут быть - размер поля, подпись поля, условие на значение, маска ввода и т.д.

*Условие на значение*. Microsoft Access не позволяет вводить данные, не удовлетворяющие этому условию. Условие на значение задается выражением, которое в общем случае состоит из операторов сравнения. Если выражение не содержит оператора, то Access будет использовать оператор “равно” (=). Также можно использовать несколько сравнений, разделенных логическими операторами OR (ИЛИ) и AND (И). Если значение, используемое в выражении, является строкой символов и содержит пробелы или специальные символы, то вся строка должна быть заключена в кавычки. Если в качестве операнда используется дата, то она заключается в символы “#”.

В следующей таблице приведены операторы сравнения, которые могут быть использованы при задании условия на значение:

|  |  |
| --- | --- |
| **Оператор** | **Назначение** |
| < | Меньше, чем |
| <= | Меньше или равно |
| > | Больше, чем |
| >= | Больше или равно |
| = | Равно |
| <> | Не равно |
| IN | Проверяет на равенство любому значению из списка; операндом является список, заключенный в круглые кавычки. |
| BETWEEN | Проверяет, что значение поля находится внутри заданного диапазона; верхняя и нижняя границы диапазона разделяются логическим оператором AND |
| LIKE | Проверяет соответствие текстового или Memo поля заданному шаблону символов |

При использовании оператора LIKE в качестве операнда можно использовать следующие символы шаблона:

|  |  |
| --- | --- |
| **Символ шаблона** | **Назначение** |
| ? | Заменяет один произвольный символ |
| \* | Заменяет любое число произвольных символов (включая нулевое количество символов) |
| # | Заменяет одну любую цифру |

Также можно задать, чтобы определенная позиция текстового или Memo поля содержала символы, указанные в списке допустимых символов, заключенном в квадратные скобки ([ ]). Если необходимо указать, что определенная позиция может содержать любой не входящий в список символ, то надо начать список с восклицательного знака (!).

Маска ввода используется, чтобы облегчить ввод форматированных данных. Microsoft Access позволяет задать маску ввода для поля с любым типом данных, кроме счетчика, поля Memo и объекта OLE. Маску ввода можно использовать для таких действий, как преобразование всех вводимых символов к верхнему регистру, добавления скобок и символов дефиса к телефонному номеру и т.д. Можно включать в маску ввода строки символов, которые будут сохраняться в этом поле.

Маска ввода состоит из трех частей, разделенных точкой с запятой. Первая часть представляет собственно маску ввода, состоящую из символов маски ввода и постоянных символов. Необязательная вторая часть указывает, требуется ли сохранить постоянные символы маски в этом поле (0 - подтверждение, 1- отказ). Необязательным третьим компонентом является символ - указатель заполняемых при вводе позиций. По умолчанию для этой цели используется символ подчеркивания (\_).

Microsoft Access использует достаточно большой набор символов для задания маски ввода. Ознакомиться с ним можно при помощи справочной системы Microsoft Access. Необходимо обратить внимание, что некоторые из них зависят от установок, заданных в разделе Установка стандарта страны Windows.

После того как все поля таблицы созданы необходимо создать первичный ключ таблицы. Для того чтобы это сделать необходимо пометить поле, которое будет являться первичным ключом, (или несколько полей) и нажать на кнопку панели инструментов Определить ключ. В подтверждение того, что ключ задан, Microsoft Access выведет в области маркировки, слева от указанного поля, символ ключа.

## Создание таблицы с помощью мастера

Одним из мощных инструментов Microsoft Access являются мастера, позволяющие пользователю создавать новые объекты по предлагаемым образцам. Microsoft Access сам строит объекты по результатам ответов пользователя на задаваемые ему в диалоговых окнах вопросы.

При создании таблицы с помощью мастера пользователю предлагается выбор из более 40 образцов таблиц, предназначенных для использования в личных или деловых целях, каждая из которых содержит соответствующий набор полей. Пользователь должен выбрать образцы полей, после чего мастер автоматически создает таблицу.

## Сортировка и поиск данных

В Microsoft Access имеется возможность сортировать данные в режиме таблицы. Самая простая сортировка может быть произведена при помощи двух кнопок на панели инструментов - сортировка таблицы в порядке возрастания или убывания значений какого-либо столбца.

Сортировка по нескольким полям может выполняться с помощью фильтра. Фильтр позволяет, также, задавать условия отбора данных из таблицы. При этом в качестве операторов для задания условий отбора можно использовать любые из описанных выше для задания Условия на значение поля таблицы.

Каждый раз, когда закрывается окно режима таблицы, введенные для фильтра параметры удаляются. Если необходимо сохранить построенный фильтр, то его можно сохранить как запрос.

## Работа с данными при помощи запроса-выборки

Использование запросов позволяет достаточно просто найти данные из нескольких связанных таблиц. Для создания запроса первым делом необходимо выбрать таблицы и запросы, которые будут являться базовыми для нового запроса. В последнем нет ничего удивительного: уже созданные запросы могут участвовать наравне с таблицами, т.к. они по сути дела ими и являются, но не хранятся все время в памяти, а формируются только во время обращения к ним.

Окно конструктора запросов содержит список базовых для этого запроса таблиц и бланк для создания запроса по образцу (QBE - Query By Example). Microsoft Access использует язык SQL для определения всех запросов. Технология QBE позволяет избежать непосредственного программирования на этом языке. Однако панель инструментов содержит кнопку, с помощью которой можно вывести на экран и отредактировать инструкцию SQL для текущего запроса.

Бланк QBE представляет собой таблицу, столбцы которой относятся к одному полю. По каждому из полей может производиться сортировка. Для задания условий отбора можно применять любые операторы, описанные для задания Условия на значение поля таблицы. При этом существует возможность создания нескольких альтернативных условий отбора с помощью ввода условий на значение не только в строке Условие отбора но и в строках или. Необходимо обратить внимание, что это не идентично использованию оператора OR внутри строки Условие отбора. Если существуют условия отбора для нескольких полей, то для любой записи, включаемой в набор записей запроса, все выражения в строке Условие отбора или в строке или должны принимать значение истина.

## Вычисляемые поля

Можно задать вычисления над любыми полями таблицы и сделать вычисляемое значение новым полем в наборе записей. При этом можно использовать арифметические операции над полями таблиц, создавать символьные константы как результат конкатенации (склеивания) текстовых полей и символьных констант и пользоваться более специфическими встроенными в Access функциями.

## Групповые операции

Иногда интерес представляет не каждая строка таблицы, а итоговые значения по группам данных (например, сумма значений одного из полей). Для вычисления в запросе итоговых значений необходимо нажать на панели инструментов кнопку Групповые операции, чтобы в бланке QBE появилась соответствующая строка.

Microsoft Access использует установку Группировка в строке Групповая операция для любого поля, которое уже занесено в бланк QBE. Это означает, что записи группируются, но итоговые значения не подводятся. Для получения итогов необходимо поменять установку Группировка на одну из групповых функций:

|  |  |
| --- | --- |
| **Функция** | **Итог** |
| Sum | Вычисляет сумму всех значений заданного поля в каждой группе. |
| Avg | Вычисляет среднее арифметическое всех значений данного поля в каждой группе. |
| Min | Возвращает наименьшее значение, найденное в этом поле внутри каждой группы. |
| Max | Возвращает наибольшее значение, найденное в этом поле внутри каждой группы. |
| Count | Возвращает число записей, в которых значения данного поля отличны от Null. |
| StDev | Подсчитывает статистическое стандартное отклонение для всех значений данного поля в каждой группе. Если в группе меньше двух строк, то результат - Null. |
| Var | Подсчитывает статистическую дисперсию для всех значений данного поля в каждой группе. Если в группе меньше двух строк, то результат - Null. |
| First | Возвращает первое значение этого поля в группе. |
| Last | Возвращает последнее значение этого поля в группе. |

Имеется также установка Выражение. Ее можно использовать, когда в строке Поле вводится выражение, в котором используется одна или несколько групповых операций.

## Использование параметров

Не всегда можно решить на этапе создания запроса, какие именно значения должен отыскать Access. Вместо этого можно включить в запрос параметр, и в этом случае перед выполнением запроса Access каждый раз будет запрашивать конкретные условия отбора.

Чтобы установить параметр, вместо того чтобы вводить конкретное значение в строку Условие отбора, необходимо ввести в нее имя параметра, заключенное в квадратные скобки ([ ]). Это имя выводится в диалоговом окне при выполнении запроса, поэтому разумно в качестве имени параметра использовать содержательную фразу.

По умолчанию Access присваивает параметрам запроса текстовый тип данных. При необходимости можно изменить тип данных параметра при помощи диалогового окна Параметры запроса.

## Перекрестные запросы

Access поддерживает специальный тип итоговых запросов, называемый перекрестным запросом. Такой запрос позволяет увидеть вычисляемые значения в виде перекрестной таблицы, напоминающей электронную. Для этого типа запросов Access добавляет в бланк QBE строку Перекрестная таблица. В этой строке для каждого поля перекрестного запроса может быть выбрана одна из четырех установок: Заголовки строк, Заголовки столбцов, Значение (выводимое в сетке перекрестного запроса) и не выводить. Для перекрестного запроса необходимо определить одно поле в качестве заголовка строки, одно поле в качестве заголовка столбца и одно поле значений. Поля, являющиеся заголовком строки или столбца, должны иметь в строке Групповая операция установку Группировка, поле значение - одну из групповых функций.

## Ограничения при использовании запроса-выборки для обновления данных

Набор записей, который создает Access при выполнении запросов, выглядит и ведет себя во многом точно так же как таблица. И действительно, во многих случаях можно вставлять строки, удалять их и обновлять данные в наборе записей. При этом Access выполнит необходимые изменения в соответствующих таблицах.

Однако в некоторых случаях нельзя понять какие изменения надо выполнить. Так нельзя изменить значение в любом вычисляемом поле или изменение, которое могло бы воздействовать на несколько строк базовой таблицы. По этой причине нельзя изменить данные в итоговом или перекрестном запросе.

# РАЗДЕЛ 3. ОРГАНИЗАЦИЯ ЗАПРОСОВ SQL

# ТЕМА 3.1. ИСТОРИЧЕСКИЕ АСПЕКТЫ. СТРУКТУРА И ТИПЫ ДАННЫХ ЯЗЫКА SQL

Язык SQL представляет собой наиболее распространенный язык управления базами данных клиент/сервер. Основное достоинство SQL заключается в том, что он унифицирован: стандартный набор инструкций SQL можно использовать в любой системе управления базами данных, которая совместима с SQL. Первый американский стандарт SQL был зарегистрирован в 1986 г. как ANSI ХЗ.135-1986. Стандартом текущей версии является ANSI ХЗ.135-1992, широко известный как SQL-92.

Примечание:

ANSI (Американский национальный институт стандартизации) — это организация, которая занимается созданием и обновлением научных и инженерных стандартов. ANSI-стандарт SQL был принят в качестве всемирного стандарта отделом ООН Международной организацией стандартизации (ISO) в 1987 г.

SQL является языком реляционных баз данных, а не языком системного программирования. SQL — язык, ориентированный на работу с множествами. Таким образом, ANSI SQL не включает ни средств управления выполнением программы (ветвлений и циклов), ни средств для создания форм или отчетов. Функции управления реализуются в языках программирования, например, xBase, С, C++ или COBOL. Однако в некоторые версии SQL, например, в Transact-SQL, используемый в Microsoft SQL-server, добавлены два оператора (IFELSE и WHILE). Разработчики новых систем управления базами данных, совместимых с ANSI SQL, вольны добавлять к языку свои расширения, но обязаны следить за тем, чтобы в новых системах поддерживался набор базовых команд ANSI. Реализация ANSI/ISO SQL никак не зависит от используемого системного языка.

ANSI SQL включает набор стандартных команд, сгруппированных по шести категориям: описание данных, выполнение запросов, манипулирования данными, управление курсором, управление транзакциями, а также администрирование или управление. В 1989 г. в исходный стандарт ANSI были добавлены инструкции для обеспечения целостности данных. Jet SQL также содержит эти операции.

**Существует три способа реализации SQL**

**Непосредственный вызов**. Инструкции SQL, передаются в СУРБД, которая создает и выводит требуемую таблицу. Примером непосредственного вызова может служить предложение SQL системы dBase IV.

**Язык модулей**. Позволяет записывать в текстовый файл инструкции SQL, которые позднее выполняются приложением. Инструкция SQL должна начинаться с ключевого слова module и определять язык программирования (language).

**Встроенный SQL**. Это наиболее распространенный метод реализации инструкции SQL генерируются системой управления или включаются как текст в команды языка этой системы. В запросах Access, создаваемых при помощи бланка запроса по образцу, в свойстве Access VBA SQL или в свойстве графика "Источник строк" (Row Source) используется встроенный SQL.

# История создания SQL

Причиной появления SQL является то, что в начале 70-х годов фирме IBIV потребовался метод, с помощью которого простые пользователи могли извлекать из базы данных и выводить нужную им информацию. Языки, которые могут применять пользователи, назвали языками четвертого поколения (4GL) или структурированным английским. Первым коммерческим результатом является язык QBE (запрос по образцу), разработанный в IBM. Начиная с конца 70-х годов, QBE используется на терминалах, подключенных к большим машинам IBM/370. При этом можно достичь требуемого результата, введя не более 80 символов, что соответствовало 100 или более строкам в COBOL или другом языке третьего поколения. В Access, dBase IV, dBase 5 и Paradox язык QBE используется для вывода требуемой информации из таблиц.

Программисты филиала IBM в Сан-Хосе, Калифорния, разработали систему System R, прототипа SQL/DS и реляционной базы данных DB2. В середине семидесятых служащий IBM доктор Е.Ф. Кодд предложил SQL (называвшийся тогда SEQUEL — Structured English Query Language) как средство извлечения информации из реляционной базы данных, модель которой он разработал в 1970 году. Разработанные на основе модели Кодда реляционные базы данных, использовавшие для извлечения и обновления информации язык SQL, стали, как и QBE, стандартами в компьютерной индустрии.

SQL стал единственным языком баз данных клиент/сервер. Сервер базы данных (нижний уровень) отвечает за хранение данных. Приложения-клиенты (верхний уровень) добавляют или обновляют данные. Кроме тог приложение генерирует инструкции SQL. При регулярной работе с базами данных значение SQL обязательно. Это также относится к разработчикам, которым требуется создавать приложения с определяемыми пользователем запросами. Кроме того, знание Jet SQL необходимо, если вы планируете пользоваться Internet Database Connector (IDC/HTX файлы), описанного главе 18.

# Построение SQL-запросов

Access поддерживает три дополнительных типа запросов, которые создаются путем введения SQL -выражений. Эти запросы нельзя создавать непосредственно в окне конструктора запроса, SQL -выражения вводятся в режиме SQL. Ниже перечислены действия, которые выполняются в Access с помощью SQL-запросов.

• Объединяющий запрос комбинирует общие поля из одной или нескольких таблиц.

• Отправляемый запрос отправляет SQL -инструкции на SQL -сервер в формате SQL базы данных этого сервера.

• Запрос, воздействующий на данные, создает или изменяет объекты непосредственно в базе данных Access.

Для создания любого из этих запросов выберите соответствующий с помощью команды Запрос - Запрос SQL (Quarry SQL Specific).

# Объединяющие запросы

Объединяющие запросы позволяют быстро комбинировать несколько таблиц, имеющих общие поля. Объединяющий запрос включает два или более SQL -выражений SELECT. Каждое выражение SELECT должно состоять из нескольких полей, расположенных в определенном порядке. Результат объединяющего запроса изменять нельзя.

# Отправляемые запросы

Отправляемые запросы пересылают SQL -команды непосредственно на SQL-сервер. Отправляемые команды должны использовать синтаксис соответствующего сервера. Перед применением таких запросов следует ознакомиться с документацией к соответствующему SQL-серверу.

Никогда не пытайтесь преобразовать отправляемый запрос в запрос другого типа, иначе Access уничтожит введенное SQL-выражение.

# Запросы, воздействующие на данные

Из всех SQL -запросов запросы, воздействующие на данные, используются реже всего. Все, что можно сделать с их помощью, доступно и при обращении к обычным средствам Access. Тем не менее, такие запросы являются весьма продуктивным средством изменения объектов базы данных. В них можно использовать SQL-выражения, перечисленные ниже.

• CREATE TABLE

• ALTER TABLE

• DROP TABLE

• CREATE INDEX

• DROP INDEX

При хорошем знании SQL можно просматривать и редактировать создаваемые Access SQL-выражения. После редактирования SQL-выражения Access автоматически фиксирует эти изменения в окне конструктора запроса.

Для просмотра SQL-выражений в меню запроса выполните команду Вид - Режим SQL (View SQL)

Для изменения существующих или создания собственных SQL-выражений изменения нужно вводить непосредственно в диалоговое окно SQL. Для увеличения количества строк нажмите <Ctrl+Enter>.

SQL-выражения можно использовать в выражениях, макросах, формах и отчетах. SQL-выражение (например, для отбора записей) можно создать прямо в окне запроса. После этого следует перейти в режим SQL, а затем нажать <Ctrl+C>, чтобы скопировать новое SQL-выражение. Затем перейдите к тому месту, где вы хотите использовать SQL-выражение, и нажмите <Ctrl+V> для его вставки.

SQL-выражение можно создавать непосредственно в диалоговом окне SQL. При написании собственного или редактирования существующего SQL-выражения после выхода из диалогового окна SQL Access автоматически обновляет окно запроса

До сих пор новые запросы создавались с помощью конструктора запроса Access. Чтобы проверить создаваемые Access SQL-выражения, можно выполнить команду Вид - Режим SQL. Один из способов научиться создавать SQL-выражения — строить запросы, а затем изучать соответствующие им запросы SQL.

В таблице представлены самые популярные команды SQL.

|  |  |
| --- | --- |
| Команда | **Назначение SQL-выражения** |
| **SELECT** | Команда/ключевое слово, с которого начинается SQL-выражение; предшествует названию поля (или полей), выбираемого из таблицы |
| **FROM** | Указывает имя таблицы или таблиц, содержащих поля, перечисленные после команды SELECT |
| **WHERE** | Команда, указывающая условие отбора или ограничение для выводимых записей; используется только в том случае, если необходимо ограничить группу отбираемых записей на основании какого-то условия |
| **ORDER BY** | Команда, указывающая порядок вывода данных |

Эти четыре основные команды SQL позволяют создавать мощные SQL-выражения, используемые в формах и отчетах Access.

# Ключевое слово DISTINCTROW

Ключевое слово DISTINCTROW в SQL-выражениях используется в Access для ограничения возвращаемых записей. Оно не используется в SQL-языке других баз данных. В Access оно служит для предотвращения вывода дублирующихся записей. Это ключевое слово работает подобно предикату DISTINCT в других реализациях SQL, но действие DISTINCT внутри запроса распространяется только на поля. DISTINCTROW проверяет записи (даже если их полей нет в выражении SELECT).

# Команда SELECT

SELECT — это первое слово запроса на выборку или на добавление. Команда SELECT используется для выбора поля (или полей), которое будет выводиться в результате.

После ключевого слова SELECT необходимо указать поля, которые нужно вывести. Если используется больше одного поля, то между полями нужно вставлять запятые:

SELECT Первое\_поле, Второе\_поле, Третье\_поле, . . .

# Предикаты SELECT

В выражениях SELECT можно использовать несколько предикатов, приведенных ниже.

• ALL

• DISTINCT

• DISTINCTROW

• TOP

Эти предикаты служат для ограничения количества возвращаемых записей. В SQL-выражении их можно использовать с командой WHERE.

Предикат ALL назначен по умолчанию. Он выбирает все записи, которые в выражении SQL удовлетворяет условию WHERE. Указывать его необязательно, поскольку он назначен по умолчанию. Предикат DISTINCT необходимо включать, когда из запроса следует исключить одинаковые записи (рассматриваются только поля, включенные в запрос). Например, при создании запроса, выводящего идентификатор покупателя и день, в который он сделал заказ, нужно использовать следующее SELECT DISTINCT [CustomerlD], [OrderDate]

Если в таблицу Orders помещено два заказа одного покупателя за один день, то в результирующей таблице будет содержаться только одна запись. Предикат DISTINCT указывает Access, что, если отобранные поля содержат одинаковые значения, нужно выводить только одну запись. Даже если на самом деле в таблице Orders есть две различные записи, то отображена будет только одна из них. Предикат DISTINCT проверяет дублирование только для полей, указанных для просмотра.

Предикат DISTINCT предназначен для исключения записей, которые содержат повторяющиеся значения в отобранных полях. Для того чтобы запись была включена в результат выполнения запроса, значения в каждом поле, включенном в инструкцию SELECT, должны быть уникальными.

DISTINCTROW— это предикат, существующий только в Access. Он работает подобно предикату DISTINCT, но с одним большим отличием: DISTINCTROW проверят совпадение в таблице или таблицах всех полей, а не только выбранных. Предикат DISTINCTROW используется для исключения записей, повторяющихся полностью. Он влияет на результат только в том случае, если в запрос включены не все поля из анализируемых таблиц. Предикат DISTINCTROW игнорируется, если запрос содержит только одну таблицу.

Если, например, какому-либо покупателю в таблице Orders соответствуют две различные записи, то при использовании в предыдущем SQL-выражении distinctrow вместо DISTINCT будут выведены обе записи. Предикат DISTINCTROW проверяет совпадение всех полей в таблицах Customers и Orders. Если содержимое каких-либо полей различно (в данном случае — идентификатор заказа), то будут выведены обе записи.

Предикат ТОР, который также характерен только для Access, ограничивает число выводимых записей, удовлетворяющих условию WHERE. Предикат TOP предназначен для возврата определенного числа записей, находящихся в начале или в конце диапазона, описанного с помощью предложения ORDER BY. Например, ТОР 10 выводит только десять первых записей, удовлетворяющих условию WHERE.

Предикат ТОР имеет один необязательный параметр PERCENT (процент), который указывает не количество первых записей, а их процентное отношение к общему числу отобранных записей.

# Использование квадратных скобок в именах полей

При задании названия поля, в котором должны быть пробелы. Access требует, чтобы это поле было заключено в квадратные скобки.

Для просмотра полей из нескольких таблиц наряду с требуемым полем нужно указывать название соответствующей таблицы. Например, предложение SQL для просмотра полей в таблицах Customers и Orders глядит следующим образом:

SELЕСТ Customers. [CompanyName], Customers. [Country], Orders. [OrderDate], lers. [Requi red Date]

Когда в Access создается запрос, названия таблиц помещаются перед названиями полей автоматически. самом деле указывать название таблицы нужно только в том случае, если в SQL-выражении несколько )лиц содержат поля с одинаковыми названиями. Например, обе таблицы — Customers и Orders — содержат поле [Customer ID]. Если в SQL-выражении нужно указать данное поле, то следует уточнить, какое полей [Customer ID] вам нужно (из таблицы Customers или из таблицы Orders):

SELЕСТ Customers. [Customer] D], [CompanyName], Country, [OrderDate], •qui redDate]

# Названия таблиц

Несмотря на то, что названия таблиц необязательны для неповторяющихся полей, их все же лучше использовать для большей ясности.

Для выбора всех полей применяется символ звездочки (\*). Для выбора всех полей из нескольких таблиц укажите название таблицы, точку (.), а затем название поля (в данном случае — звездочку).

**ТЕМА 3.2. ОПЕРАТОРЫ ЯЗЫКА SQL**

# Команда FROM в SQL-выражении

Команда FROM, как следует из ее названия, указывает таблицы (или запросы), содержащие поля в выражении SELECT. Эта команда указывает Access, где искать записи. При работе с одной таблицей, как в приведенном примере, команда FROM просто указывает имя таблицы:

SELECT [CoinpanyName], Country, FROM Customers

При работе с несколькими таблицами в команде FROM можно указывать, какие данные выбираются. В команде FROM устанавливается объединение между несколькими таблицами, которые будут соединены в результирующей таблице. Объединения бывают трех типов:

• I NNER JOIN. ..ON

• RIGHT JOIN...ON

• LEFT JOIN.. .ON

INNER JOIN. . .ON используется для указания традиционного объединения Access.

LEFT JOIN и RIGHT JOIN работают почти так же, с той лишь разницей, что вместо внутреннего объединения они указывают внешнее.

# Команда WHERE в SQL-выражениях

Команда WHERE в SQL-выражениях служит для указания условий. Выражение SQL можно начать, к примеру, с команды WHERE:

WHERE (Customers. [Customer I D]="EASTC")

Условие WHERE может быть любым допустимым выражением. Оно может быть простым выражением, состоящим из одного условия (как в приведенном примере), или же сложным выражением, включающим несколько критериев.

В SQL-выражении команда WHERE должна следовать за командой FROM.

# Команда ORDER BY

Команду ORDER BY можно использовать для сортировки запроса. С помощью ORDER BY данные сортируются по возрастанию или по убыванию. Например, для указания порядка сортировки по полю CustomerlD следует ввести следующее:

ORDER BY Customers. [CustomerlD];

# Окончание SQL-выражения

Поскольку SQL-выражение может состоять из 64 000 символов, необходимо обозначить его конец. SQL-выражение необходимо оканчивать точкой с запятой (;).

Access забывает о том, что вы должны использовать завершающую точку с запятой. Если точка с запятой не будет помещена в конец запроса, то запрос выполнится так, как будто точка с запятой в конце есть.

Если точка с запятой случайно окажется внутри SQL-выражения, то Access выдаст сообщение об ошибке и попытается указать, где она произошла.

При использовании SELECT, FROM, WHERE и ORDER BY можно создавать различные SQL-выражения для просмотра и вывода данных из таблиц. Например, можно построить SQL-выражение, выполняющее действия, перечисленные ниже.

1. Выбор полей CompanyName, Country, OrderlD и OrderDate.

2. Объединение таблиц Customers и Orders по полю Customer] D.

3. Вывод только тех записей, в поле CustomerlD которых содержится EASTC.

4. Сортировка данных по полю OrderDate. Соответствующее SQL-выражение должно выглядеть следующим образом:

SELECT [CompanyName], Country, [OrderlD], [OrderDate] FROM Customers INNER JOIN Orders ON Customers. [CustomerlD] = \_ Orders. [CustomerlD] WHERE [CustomerlD] = "EASTC" ORDER BY Orders. [OrderDate];

Приведенная ниже информация о SQL-выражениях может служить описанием процесса создания SQL-выражений в Access 97. В SQL-выражениях можно также использовать другие команды. SQL относительно легок для понимания и использования и является неоценимым помощником в создании запросов и их использовании.

# Создание запросов-действий при помощи SQL

# Обновление, удаление и вставка данных из другой таблицы при помощи запросов-действий

Любые изменения исходных данных в Microsoft Access можно проделать даже в режиме таблицы, но на самом деле это справедливо, если работа производится с одной или с достаточно небольшим набором записей. Если нужно произвести одинаковые действия со множеством записей, то работа с каждой записью отдельно может оказаться невыполнимой.

Чтобы избежать этого в Microsoft Access существуют такие объекты как запросы-действия, которые позволяют найти необходимые записи и автоматически произвести в них необходимые изменения. Для изменения данных уже существующих таблиц Access имеет три вида запросов-действий:

Запрос-обновление. Данный вид запросов позволяет произвести изменение значений одного или нескольких полей выбранных записей. При этом новые значения могут получаться как на основе старых данных, так и каким-либо другим способом.

Запрос на удаление. Как правило не имеет смысла всегда хранить все собранные в базе данные. Имеет смысл со временем заменять часть исходной информации итоговой и затем удалять ненужные данные. Именно для этих целей и служит данный вид запросов.

Запрос на добавление. Используя данный вид запросов, можно скопировать выбранные записи и поместить их в другую таблицу.

В момент создания любого запроса Access всегда по умолчанию создает запрос-выборку. При создании запросов-действий необходимо сначала создать запрос-выборку, используя условия отбора, позволяющие выбрать те записи, над которыми необходимо произвести соответствующие действия. После этого при помощи меню или панели инструментов можно преобразовать полученный запрос-выборку в один из запросов-действий одним щелчком мыши.

# Запрос-обновление

При преобразовании запроса-выборки в запрос-обновление изменяется заголовок окна запроса и в бланке QBE появляется строка Обновление. Эта новая строка бланка используется для того, чтобы задать, как именно изменять данные.

С помощью запроса-обновления можно за один раз менять значения нескольких полей, включив их в бланк QBE и определив выражения, используемые для обновления новых записей. При этом для вычисления новых значений некоторого поля можно использовать значения других полей.

Когда Access готов приступить к обновлению записи в базовой таблице или запросе, он сначала делает копию исходной записи, затем применяет заданные выражения к исходной записи и помещает результат в копию. После этого он меняет содержимое БД, переписывая модифицированную копию в таблицу. Так как обновления выполняются в копии исходной записи, то нет необходимости специально обеспечивать какую-либо проверку актуальности данных при изменении значений нескольких полей с использованием для вычисления новых значений данных из других полей, подлежащих изменению (например, можно поменять местами данные двух полей).

# Инструкция UPDATE

Инструкцию UPDATE можно использовать для создания запроса на обновление, который изменяет значения в полях. Синтаксис:

UPDATE имя таблицы

SET имя столбца = новое значение

[WHERE условия отбора]

В указанной таблице или запросе происходит обновление выбранных строк в соответствии со значением или выражением, указанным как новое значение. Если присутствует предложение WHERE, то обновляются только строки, которые удовлетворяют условию отбора

# Запрос на удаление

Как правило, этот тип запросов-действий используется для удаления из таблиц устаревших данных. Для этого на основе информации о системной дате и о дате в конкретных записях производится выборка данных, которые должны быть удалены, а затем их удаление.

Очевидно, что такой запрос будет использоваться регулярно. Поэтому имеет смысл, чтобы запрос содержал параметры, с помощью которых пользователь мог бы указать период за который все данные должны быть удалены. В этом случае не нужно будет в определение запроса вносить изменения каждый раз перед его использованием.

# Инструкция DELETE

Эта инструкция удаляет одну или несколько строк из таблицы или запроса. Предложение WHERE не является обязательным. При его отсутствии из таблицы или запроса, которые указаны в предложении FROM, будут удалены все строки. При использовании предложения WHERE к каждой строке таблицы или запроса будет применено условие отбора и будут удалены только выбранные строки. Синтаксис инструкции:

DELETE поля, по которым проводится отбор

FROM имя таблицы или запроса

WHERE условия отбора

Запрос на удаление записей удаляет целые записи, а не содержимое отдельных полей. Для удаления значений из определенного поля создайте запрос на обновление, который заменит эти значения пустыми (Null) значениями.

# Запрос на добавление

Используя запрос на добавление, можно скопировать выбранные записи и поместить их в другую таблицу. Удобно также использовать запрос на добавление, чтобы включить в БД часть информации из другой БД.

При создании такого запроса необходимо задать таблицу из которой берутся данные и таблицу куда они направляются. Access вводит в бланке QBE новую строку Добавление. Она показывает соответствие между копируемыми полями и полями, куда поступают данные. Если в таблицах имеются поля с совпадающими именами Access автоматически сопоставляет их друг другу и вносит в бланк QBE.

# Инструкция INSERT

Инструкцию INSERT INTO можно использовать для создания запроса на добавление записей, который добавляет записи из таблицы или запроса из текущей базы данных в конец другой таблицы или запроса в текущей или другой базе данных. Синтаксис:

Запрос на добавление одной записи:

INSERT INTO таблица-адресат [список полей]

VALUES список значений

Запрос на добавление нескольких записей:

INSERT INTO таблица-адресат [список полей]

SELECT список полей

FROM выражение

Где таблица-адресат представляет собой таблицу, в которую вносятся новые записи, список значений - значения, которые следует добавить в указанные поля новой записи. Предложение FROM определяет таблицу, из которой следует скопировать записи, а предложение SELECT указывает имена полей из этой таблицы, которые должны быть добавлены.

# Создание новой таблицы с помощью запроса

Иногда нужно сохранить извлекаемые с помощью запроса-выборки данные в реально существующей таблице. Например, если почему-либо приходится снова и снова выполнять один и тот же запрос, а исходные данные для него не меняются. Если для выполнения запроса использовалось несколько таблиц, то при использовании таблицы вместо запроса доступ к данным будет существенно ускорен. При превращении запроса-выборки в запрос на создание новой таблицы необходимо задать имя создаваемой таблицы и базу данных, в которой эту таблицу следует сохранить (по умолчанию в текущей БД).

# 

# Инструкция SELECT ... INTO

Данная инструкция позволяет создавать новую таблицу, имя которой задается в качестве параметра имя новой таблицы.

SELECT [предикат] список полей [AS псевдонимы полей]

INTO имя новой таблицы

FROM список таблиц [IN для таблиц внешних баз данных]

[WHERE условие отбора строк ]

[GROUP BY группируемые поля ]

[HAVING условия отбора групп ]

[ORDER BY поля для сортировки и условия их сортировки ]

[WITH OWNERACCESS OPTION]

Значение остальных параметров ничем не отличаются от обычной инструкции SELECT.

# Ошибки при выполнении запросов-действий

Перед тем как внести соответствующие изменения в БД, Microsoft Access анализирует запрос и те данные, которые изменяются при выполнении запроса-действия. Если обнаруживается ошибка, Access предоставляет пользователю возможность отменить операцию до окончательного выполнения запроса-действия. При этом различаются следующие ошибки:

* Дубликаты первичного ключа. Возникает при попытке добавить или изменить некоторую запись в таблице, в результате чего появляется дублирующее значение первичного ключа.
* Ошибки преобразования данных. Возникает при добавлении данных в таблицу, когда тип полей-получателей не совпадает с типом полей-источников (и данные полей-источников не могут быть соответствующим образом преобразованы).
* Заблокированные записи. Access не позволяет изменять данные в таблице, к которой в данный момент имеют доступ другие пользователи (в режиме коллективного доступа). После сообщения о данной ошибке следует попробовать снова выполнить операцию через некоторое время.
* Нарушение условий на значение. Возникает в случая нарушения условий на значение удаляемых или обновляемых полей.

Следует обратить внимание, что при усечении добавляемых в текстовые или поля Memo данных (при преобразовании их форматов) Access не выдает никаких предупреждений. По этому необходимо следить, что в таких ситуациях поля-приемники должны иметь достаточный для хранения поступающих данных размер.

# Инструкция TRANSFORM

Данная инструкция порождает перекрестный запрос. Инструкция TRANSFORM имеет следующий синтаксис:

TRANSFORM функция

инструкция SELECT

PIVOT выражение

Функция определяет значения, которые должны появиться в ячейках перекрестной таблицы, а PIVOT выражение определяет столбец или выражение, значения которого будут использоваться в качестве столбца.

**Контрольные вопросы**

1. Что такое запрос ?
2. Назовите элементы окна конструктора запросов.
3. Что такое бланк QBE ?
4. Где записываются критерии условия выбора для запроса?
5. Как удалить таблицу из запроса?
6. Как удалить поле из запроса?
7. Перечислите основные операторы, используемые в запросе.
8. В каких случаях используется оператор OR?
9. В каких случаях используется оператор AND?
10. В чем различие между операторами OR и AND ?
11. Назначение итоговых запросов.
12. Назначение построителя выражений.
13. Как вычислить сумму значений заданного поля?
14. Как осуществить сортировку записей в запросе?
15. Какие виды вычислений можно произвести в итоговых полях?
16. Типы запросов в Access. Как выбрать тип запроса?
17. Как создать запрос на обновление части данных в таблице?
18. Как удалить часть данных в таблице?
19. Какие проблемы могут возникнуть при использовании запросов удаления?
20. Как создать таблицу для постоянного хранения результатов определенного запроса?
21. Как посмотреть результат запроса?
22. Как можно изменить значение выбранного поля по всем записям таблицы, удовлетворяющим определенным условиям?

**РАЗДЕЛ 4. ОРГАНИЗАЦИЯ ИНТЕРФЕЙСА С ПОЛЬЗОВАТЕЛЕМ**

**ТЕМА 4.1. ПОНЯТИЕ ОБЪЕКТА, СВОЙСТВА И ХАРАКТЕРИСТИКИ ОБЪЕКТА. СОЗДАНИЕ ЭКРАННОЙ ФОРМЫ: СВОЙСТВА, СОБЫТИЯ И МЕТОДЫ**

# Назначение и область применения форм

Формы в Microsoft Access являются основным средством организации интерфейса пользователя. Обычно формы применяют для достижения следующих целей.

# Ввод, вывод и редактирование данных

Это наиболее распространенное применение форм. Формы обеспечивают вывод на экран данных в удобном для этого представлении. С их помощью значительно упрощается внесение изменений, добавление и удаление данных из БД. В форме можно сделать доступными только для чтения все данные или некоторую их часть, автоматически выбирать информацию из других связанных таблиц и помещать ее в форму, вычислять выводимые в форме значения, скрывать или, наоборот, выводить некоторые данные в зависимости от значений других полей записи, либо от выбранных пользователем параметров. Можно также создавать формы, предназначенные только для ввода новых значений.

# Управление ходом выполнения приложения

Чтобы автоматизировать вывод определенных данных или выполнение некоторой последовательности действий, можно создать формы для работы с макросами или функциями Microsoft Access Basic. Для запуска макросов и процедур Access Basic можно создать специальные элементы управления – командные кнопки. С помощью макросов и процедур можно открывать другие формы, выполнять запросы, фильтровать выводимые на экран данные, распечатывать отчеты и выполнять другие необходимые при работе приложения действия.

# Вывод сообщений

Формы могут предоставлять вспомогательную информацию о работе приложения или выполняемых действиях. При помощи такого механизма можно, например, выводить сообщение об ошибке в ответ на какие-либо недопустимые действия пользователя.

# Печать информации.

Несмотря на то, что для печати информации более приспособлено такое средство как отчеты, иногда удобно распечатать информацию так, как она представлена в форме. Это имеет смысл по причине того, что Microsoft Access позволяет определить один набор параметров для вывода формы на экран, а другой – для ее печати.

# Формы и их основные элементы

# Заголовок, область данных и область примечаний

Обычно информация, которую необходимо вывести из базовой таблицы или запроса, размещается в области данных, расположенной в центре окна формы. Чтобы вывести на экран информацию или элементы управления, которые не будут меняться от записи к записи, можно добавить в верхнюю часть окна формы заголовок, а в нижнюю часть – область примечаний. Области примечаний и заголовок

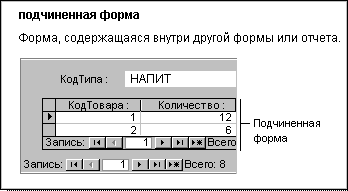
отличаются от остальной части окна – области данных, только тем, что не двигаются относительно окна при перемещении по данным в случае ленточной или многострочной формы, что означает возможность использования в этих областях любых элементов управления и ввода/вывода в форму.

**Многостраничные и ленточные формы.**

Если запись, которую необходимо вывести на экран, содержит большой объем информации, то имеет смысл создать многостраничную форму. Такая форма содержит полосу прокрутки, с помощью которой можно свободно перемещаться по записям, чтобы их все, в том числе и не умещающиеся в данный момент на экране.

Можно создать форму другого типа, которую удобно использовать для просмотра списка записей с небольшим числом полей данных. Это ленточная форма. В отличии от простой формы, которая показывает только одну запись, в ленточной форме отформатированные записи выводятся одна за другой.

Вообще говоря, для многостраничной формы слово “создать” не совсем подходит, т. К. Access сам добавит полосу скроллинга, если все поля формы не уместились в заданных для нее размерах окна. В отличие от многостраничной формы для создания ленточной формы нужно установить значение Ленточная форма для параметра формы Режим по умолчанию.

**Подчиненные формы.**

Если нужно выводить данные из нескольких таблиц и редактиро­вать данные более чем одной таблицы, то требуется более сложная форма. Microsoft Access позволяет создавать подчиненные формы и ус­танавливать связь с главной формой. Это синхронизирует данные, выводимые в обеих формах. В форму можно внедрять другие формы до трех уровней вложенности. Начинать всегда следует с создания наиболее вложенной формы и продолжать работу, используя это правило, чтобы постепенно подойти к главной форме. Для создания несвязанного элемента управления Под­чиненная форма нужно воспользоваться соответствующей кнопкой на панели инструментов конструктора.

Чтобы связать главную и подчиненную формы, нужно установить три свойства элемента управления Подчиненная форма. Необходимо задать Объект-источник- имя формы, которая будет использоваться как подчиненная, Основное поле – имя элемента главной формы, значе­ние которого определяет, что именно выводится в подчиненной форме, и Подчиненное поле – поле подчиненной формы данные в котором син­хронизируются с основным полем.

**Контрольные вопросы:**

1. С какой целью создаются формы? Назовите виды форм MS Access.

2. Назовите инструменты создания форм.

3. Назовите виды автоформ.

4. Перечислите основные разделы форм и их назначение.

5. Для чего используется раздел **Данные**в форме?

6. Как разместить в форме элемент управления?

7. Как используется при создании форм **Список полей**?

8. Как изменить порядок обхода элементов в форме?

9. Как построить форму по двум связанным таблицам?

10. Назовите основные элементы диаграмм.

11. Как построить диаграмму, используя режим Мастера диаграмм?

**ТЕМА 4.2. ХРАНИМЫЕ ПРОЦЕДУРЫ И ТРИГГЕРЫ. ОБЕСПЕЧЕНИЕ ДОСТОВЕРНОСТИ, ЦЕЛОСТНОСТИ И НЕПРОТИВОРЕЧИВОСТИ ДАННЫХ. КАСКАДНЫЕ ВОЗДЕЙСТВИЯ. МАКРОСЫ**

Триггеры представляют собой методы, с помощью которых можно обеспечивать целостность базы данных даже в том случае, если она используется множеством приложений.

Триггер - это специальный тип хранимой процедуры, которая автоматиче­ски выполняется при каждой попытке изменить защищаемые ей данные. Триггеры обеспечивают целостность данных, предотвращая их несанкцио­нированное или неправильное изменение.

Допустим, что в базе данных есть таблицы, связанные через поле Surname. Например, это могут быть таблица клиентов предприятия и их заказов. Ра­зумно определить триггер, который при каждой попытке удалить запись клиента проверит наличие у него заказов и позволит удалить эту запись только при их отсутствии. Конечно, подобную задачу можно решить при помощи средств декларативной ссылочной целостности. Однако при помо­щи триггеров можно создавать значительно более сложные рабочие правила. Можно создать триггер, который при каждом добавлении записи в таблицу заказов анализирует предыдущие заказы этого же клиента и определяет при­емлемый срок оплаты этого заказа.

Триггеры не принимают параметров и не возвращают значений. Они вы­полняются неявно. То есть триггер запускается только при попытке измене­ния данных.

Триггеры могут иметь до 32 уровней вложенности. Вложенные триггеры ра­ботают следующим образом: пусть при создании записи о новом заказе триггер добавляет информацию в таблицу неоплаченных счетов. При этом выполняется другой триггер, который проверяет, имеет ли клиент просро­ченные неоплаченные счета и, если они есть, триггер выводит сообщение об этом. В этом примере один триггер обновляет таблицу, вызывая при этом выполнение другого триггера.

По умолчанию все триггеры (INSERT, DELETE, UPDATE) срабатывают сразу по­сле выполнения операции изменения данных. Эти триггеры относятся к ти­пу AFTER (после). Начиная с SQL Server 2000 появилась еще одна группа триггеров - INSTEAD OF (вместо), которые выполняются вместо оператора, предполагающего изменение данных.

С точки зрения быстродействия триггеры не имеют никаких преимуществ. Выполнение триггера связано с постоянными обращениями к различным таблицам. Соответственно его работа быстрее, если используемые таблицы находятся в оперативной памяти, и медленнее - если данные считываются с диска.

Триггер является частью транзакции, следовательно, если триггер терпит неудачу, отменяется вся транзакция. И наоборот, если какая-то часть тран­закции не удалась, то и триггер будет отменен.

В своей работе триггеры используют таблицы Inserted и Deleted. Это логиче­ские таблицы, они постоянно находятся в оперативной памяти и имеют туже структуру, что и таблица, для которой создан триггер. Каждая добавлен­ная к защищаемой триггером таблице строка добавляется и в таблицу Inserted. Обновление производится почти так же, как и удаление с после­дующей вставкой. Когда строка обновляется, старая строка записывается в таблицу Deleted, затем обновленная строка записывается в базовую табли­цу и в таблицу Inserted.

**Создание триггера**

Триггер создается оператором CREATE TRIGGER. Рассмотрим его синтаксис:

CREATE TRIGGER [владелец.] имя\_триггера

ON [владелец.] имя\_таблицы │имя\_представления

FOR (AFTER│ INSTEAD OF} (INSERT│UPDATE I DELETE}

[WITH ENCRYPTION]

AS оператор\_SQL

Таблица может иметь произвольное количество триггеров любых типов (INSERT, UPDATE или DELETE). ПО умолчанию триггер выполняется после из­менения данных, однако, если указать параметр INSEAD OF, то создается триггер, выполняющийся вместо изменения данных.

Каждая операция (INSERT, UPDATE и DELETE) может вызывать выполнение произвольного количества триггеров. С единственным ограничением ­имена триггеров, вызываемых одной операцией, должны быть уникальными. Изменить триггер можно, удалив его и создав заново в другом виде или при помощи оператора ALTER TRIGGER. При удалении таблицы, имеющей триг­геры, все они также удаляются.

При создании триггеров необходимо придерживаться следующих правил:

* Триггеры создаются для поддержания целостности данных, ссылочной целостности и рабочих правил.
* Нельзя создавать триггеры для временных таблиц. Однако триггеры могут к таким таблицам обращаться так же, как и к представлениям.
* Триггер не может возвращать результирующих наборов данных. Это зна­чит, что к использованию оператора SELECT при его создании нужно подходить крайне осторожно. Обычно в этих случаях используется оператор SELECT с директивой IF EXISTS.
* С помощью опции WITH ENCRYPTION исходный код триггера, хранящийся в таблице syscomments, можно зашифровать.
* Операторы WRITETEXT не инициализируют триггеры. Они используются для изменения данных типа Text или Image, а эти изменения не заносят­ся в журнал транзакций.
* В триггерах нельзя использовать следующие операторы: все операторы CREATE, все операторы DROP, ALTER TAВLE, ALTER DATAВASE, TRUNCATE TABLE, GRANT и REVOKE, RECONFIGURE, LOAD DATAВASE или TRANSACTION, UPDATE STATISTICS, SELECT INTO и все операторы DISK.
* Операторы отмены транзакций, входящие в состав триггера, могут стать причиной непредсказуемого поведения операторов вызывающей программы.

Триггеры также применяются для поддержки правил. Конечно, для этого час­то достаточно использования ограничений ANSI, значений по умолчанию или пользовательских типов данных, однако при необходимости обращения к другим таблицам триггеры оказываются совершенно незаменимыми. Кроме того, при помощи триггеров реализуется механизм ссылочной целостности. Нужно отметить, что триггеры изначально были разработаны именно для этой цели. Они особенно эффективны при каскадном удалении и обновлении дан­ных. При изменении данных условия триггеров проверяются в последнюю очередь, а в начале проверяются ограничения. То есть если условие ограниче­ния нарушено, то операция отменяется, и триггер не срабатывает.

Отдельное внимание стоит уделить триггерам типа INSTEAD OF. Если триггер создается с этой опцией, то код триггера выполняется не после заданной пользователем (или удаленной программой) команды, а вместо нее. Напри­мер, разумно использовать триггеры INSTEAD OF для сообщений о невоз­можности удаления какого-либо объекта. Разумеется, эту функцию можно реализовать и с помощью триггера AFTER, однако в этом случае придется отменять уже проделанную операцию, что неприемлемо для высоко загру­женных систем, где ведется строгий контроль производительности.

**Хранимые процедуры. Назначение**

Хранимая процедура - это последовательность компилированных операторов Тгаnsасt - SQL, хранящихся в системной базе данных SQL Server. Хранимые процедуры предварительно откомпилированы, поэтому эффективность их выполнения выше, чем у обычных запросов. Хранимые процедуры работают непосредственно на сервере и хорошо укладываются в модель клиент- сервер.

Существует два вида хранимых процедур: системные и пользовательские.

Системные хранимые процедуры предназначены для получения информации из системных таблиц и выполнения различных служебных операций и осо­бенно полезны при администрировании базы данных. Их имена начинаются с sp\_ (stored procedure).

Пользовательские хранимые процедуры создаются непосредственно разработ­чиками или администраторами базы данных.

Полезность хранимых процедур определяется в первую очередь высокой скоростью их выполнения. Кро­ме того, они являются средством систематизации часто выполняемых операций. При выполнении в первый раз хранимой процедуры можно выделит ряд этапов.

* Процедура разбивается на отдельные компоненты лексическим анализатором выражений.
* Компоненты, ссылающиеся на объекты базы данных (таблицы, индексы представления и т. п.), сопоставляются с этими объектами с предвари тельной проверкой их существования. Этот процесс носит название *разрешение ссылок*.
* В системной таблице syscomments сохраняется исходный текст процедуры, а в таблице sysobjects - ее название.
* Создается предварительный план выполнения запроса. Этот предварительный план называется нормализованным планом или деревом запроса и хранится в системной таблице sysprocedures.
* При первом выполнении хранимой процедуры дерево запроса считывается и окончательно оптимизируется. Выполняется ранее созданный План процедуры.

Такая схема дает возможность при повторных вызовах не тратить время на синтаксический анализ, разрешение ссылок и компиляцию дерева запросов.

А при последующих вызовах выполняется только пятый шаг. Причем план хранимой процедуры после первого выполнения содержится в быстродейст­вующем процедурном кэше. Это значит, что во время вызова процедуры скорость его считывания будет очень высока.

Использование хранимых процедур имеют еще ряд дополнительных Пре­имуществ.

* Хранимые процедуры позволяют выделять правила в отдельную структуру. В дальнейшем эти правила используются многими приложениями, образуя устойчивый к ошибкам интерфейс данных. Выгода такого подхо­да состоит в том, что можно осуществлять изменение правил только для отдельной части объектов базы данных, а не для всех ее приложений.
* Использование хранимых процедур значительно повышает производи­тельность запросов, однако наибольшей ее прирост достигается при вы­полнении многократно повторяющихся операций, когда план запроса постоянно хранится в системном кэше.
* Хранимые процедуры могут принимать аргументы при запуске и возвра­щать значения (в виде результирующих наборов данных).
* Хранимые процедуры могут запускаться по расписанию (в режиме авто­матического выполнения), задаваемому при запуске SQL Server.
* Хранимые процедуры используются для извлечения или изменения дан­ных в любое время.
* Хранимые процедуры, в отличие от триггеров, вызываются явно. То есть при непосредственном обращении к процедуре из приложения, сцена­рия, пакета или задачи.

Хранимые процедуры - мощное средство обработки данных. Системные хранимые процедуры играют очень важную роль в администрировании и поддержке базы данных. Пользовательские хранимые процедуры при меня­ются при решении практически любых задач. Кроме того, пользователь мо­жет получить право выполнения хранимой процедуры, даже если он не име­ет права доступа к объектам, к которым обращается процедура.

**Макросы, их создание**

# Общие сведенья о макросах

**Макросом** называют список выполняемых инструкций Microsoft Access, которому присвоено имя. Каждую из инструкций называют макрокомандой. При запуске макроса макрокоманды выполняются в порядке их расположения в списке; при этом используются объекты или данные, указанные в аргументах макрокоманд. Макросы позволяют автоматизировать выполнение повторяющихся действий и расширить функциональные возможности базы данных.

**Макрокоманда -** основной строительный блок макроса: самодостаточная инструкция, которая может быть объединена с другими макрокомандами, чтобы автоматизировать выполнение задачи.

В MS Access можно определить макрос, выполняющий практически те же действия, которые можно реализовать нажатием на клавиши или кнопку мыши. Основным преимуществом макросов в Access является то, что они могут вызываться в ответ на многие виды событий. Событие может являться изменение данных, открытие или закрытие формы или отчета

Аргумент макрокоманды - дополнительные сведенья, необходимые некоторым макрокомандам, например, имя обрабатываемого объекта или условие, при котором данная макрокоманда должна быть выполнена.

Microsoft Access предоставляет различные типы макрокоманд для автоматизации приложения:

1. Вы можете использовать макросы для открытия любой таблицы, запроса, формы или отчета в любом режиме. Вы можете также использовать макрос для закрытия любой открытой таблицы, запроса формы или отчета.
2. Вы можете использовать макросы для открытия отчета в режиме предварительного просмотра или для непосредственного вывода отчета на принтер. Вы также можете вывести данные из отчета в файл в формате RTF (расширенный текстовый формат) или файл Microsoft Excel (XLS).
3. Макрокоманды можно использовать для выполнения запроса-выборки или запроса-действия. Можно задавать параметры запроса с помощью элементов управления любой открытой формы.
4. Макрокоманды можно использовать для выполнения действия, зависящего от значения величин в база данных, форме или отчете. Макросы можно использовать для запуска других макросов или процедур Microsoft Access Basic. Можно прервать выполнение текущего макроса или всех макросов, отменить событие, которое запустило макрос, или даже выйти из приложения.
5. Макросы можно использовать для установки любого значения элемента управления формы или отчета. Можно также эмулировать действия клавиатуры и передавать вводимые с клавиатуры данные в системные диалоговые окна.
6. Макросы можно использовать для построения фильтра, для перехода к любой записи, для поиска данных в базовой таблице или базовом запросе формы.
7. Макросы можно использовать для выполнения любой команды любого меню Access.
8. Макросы можно использовать для перемещения, изменения размеров, сворачивания или восстановления любого окна внутри рабочего окна Access. С помощью макросов можно передать фокус любому из окон или любому элементу управления внутри окна.
9. Макросы можно использовать для вывода на экран информационных сообщений и звуковых сигналов для привлечения внимания к вашим сообщениям. Также с помощью макросов можно отключать системные сообщения при выполнении запросов - действий.
10. Макросы можно использовать для перемещения любого из объектов базы данных. Можно сделать копию объекта либо скопировать его в другую базу данных.
11. Макросы можно использовать для запуска других приложений как в среде MS Windows так MD DOS, а так же осуществлять обмен данными с приложением, используя механизм DDE или буфер обмена.

# Общие сведенья о модулях

Как и для других объектов базы данных, в Microsoft Access имеется набор инструментов для создания необходимой программы Microsoft Access Basic. Имеется набор инструментов для отладки и анализа, созданных программ, который поможет вам убедиться, что разработанная вами программа функционирует должным образом.

Все написанные программы хранятся в модулях. База данных может содержать два вида модулей: общие модули, являющиеся объектами базы данных, и модули форм и отчетов, которые являются частью этих объектов.

*Общие модули*

Общие модули используются для хранения процедур, которые будут выполняться и запросов или из нескольких форм или отчетов приложения. Процедуры общего модуля могут быть вызваны из любого места в приложении.

*Модули, связанные с формами и отчетами.*

Каждая форма или отчет содержат встроенный модуль. Можно создать специальные процедуры внутри модуля, связанного с формой или отчетом, которые будут доступны только из данной формы или отчета. Чтобы отредактировать связанный модуль, необходимо открыть соответствующую форму или отчет в режиме конструктора и затем щелкнуть по кнопке Программы на панели инструментов.

Модули форм или отчетов имеют два важных преимущества перед общими модулями:

программа необходимая для автоматизации действий в некоторой форме или отчете, храниться в месте с данной формой или отчетом. Не возникает необходимости запоминать имя того модуля, который используется для хранения процедур обработки событий определенной формы или отчета.

Access загружает общие модули базы данных при ее открытии, а встроенный модуль формы или отчета загружается только при открытии соответствующего объекта. В связи с этим модули и формы или отчета расходуют память приложения только в том случаи, когда используется соответствующая форма или отчет.

Когда следует использовать модули.

Хотя макросы и являются мощным средством, все-таки можно назвать некоторые задачи, которые нельзя решить с помощью макросов.

Целесообразно использовать модули вместо макросов во всех перечисленных ниже случаях, когда:

1. необходимо отслеживать и обрабатывать ошибки приложения
2. требуется проводить обработку событий с передачей или получением параметров;
3. во время работы приложения нам нужно создавать новые создавать новые объекты (таблицы, запросы , формы или отчеты
4. приложение должно взаимодействовать с другими приложениями Windows, используя динамический обмен данными (DDE);
5. когда необходимо непосредственно использовать системные функции Windows;
6. нужно поместить часть программ в библиотеку;
7. необходимо обрабатывать отдельные записи, а не наборы записей.

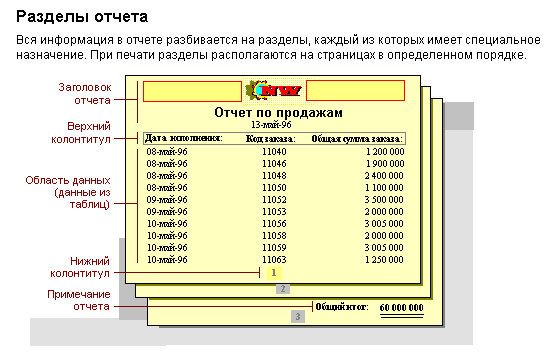
**Заключение**

На занятии были рассмотрено назначение и возможностях форм и отчетов, а также как правильно использовать эти средства в своих приложениях. Рассмотрены такие средства автоматизация приложений, как макросы и модули.

**ТЕМА 4.3. ФОРМИРОВАНИЕ И ВЫВОД ОТЧЕТОВ**

# Назначение и возможности отчетов.

Отчеты предназначены для того, чтобы выбирать данные из нескольких таблиц, производить над ними вычисления, подводить итоги и выводить их на экран или печать. Они имеют широкие возможности для группировки, а так же вычисления промежуточных и общих итогов для больших наборов данных.

Отчеты и формы имеют много общих конструктивных черт. Как и формы, отчеты также имеют заголовки и области данных, а так же есть возможность создавать подчиненные отчеты и вставлять их в другие отчеты.

Отчет может быть создан на основе таблицы, запроса или инструкции SQL. Для организации связи между отчетом и его исходными данными применяются элементы управления. В качестве элементов управления используются поля, Содержащие имена и числа, надписи для заголовков, декоративные линии для графического оформления отчета. Возможна так же вставка объектов OLE.

Отчеты могут быть открыты в трех режимах:

* в режиме конструктора
* в режиме предварительного просмотра
* в режиме просмотра образца.

Режим конструктора используют для создания отчета или изменения структуры существующего отчета.

Режим предварительного просмотра позволяет просматривать данные в том виде, в котором они будут размещаться на печатных страницах.

В режиме просмотра образца выводятся основные элементы макета отчета с образцами данных, демонстрирующих представление данных в отчете.

Пользователь имеет возможность разработать отчет самостоятельно или создать отчет с помощью мастера. Мастер по разработке отчетов MS Access выполняет всю рутинную работу и позволяет быстро разработать отчет. После вызова мастера выводятся окна диалога с приглашением ввести необходимые данные, и отчет создается на основании ответов пользователя. Мастер окажется полезным даже для опытных пользователей, так как позволяет быстро разработать макет, служащий основой создаваемого отчета. После этого можно переключиться в режим конструктора и внести изменения в стандартный макет.

**ЗАЩИТА БАЗЫ ДАННЫХ**

Microsoft Access обеспечивает два традиционных способа защиты базы данных: установку пароля, требуемого при ее открытии, и защиту на уровне пользователя, позволяющую ограничи­вать ту часть БД, к которой пользователь будет иметь доступ или сможет изменять. Кроме того, можно удалить изменяемую программу Visual Basic из базы данных, чтобы предотвратить изменения струк­туры форм, отчетов и модулей, сохранив БД как файл MDE.

Простейшим способом защиты БД является установка пароля. В этом случае при каждом открытии базы данных будет появлять­сядиалоговое окно, в которое требуется ввести пароль, и только те пользователи, которые введут правильный пароль, смогут от­крыть эту базу данных. Данный способ достаточно надежен для БД, которая совместно используется небольшой группой пользо­вателей или на автономном компьютере, установка пароля обыч­но оказывается достаточной.

Наиболее гибким и распространенным является способ защи­*ты* базы данных на уровне пользователей, который подобен спо­собам защиты, используемым в большинстве сетевых систем. В этом случае от пользователей требуется идентифицировать себя и ввес­ти пароль, когда они запускают Мicrosоft Access.

С помощью средств защиты можно указать, какие операции по обработке объектов базы данных разрешается выполнять конкрет­ному пользователю или группе пользователей. При этом о каждом пользователе или группе пользователей ведутся учетные записи с указанием прав доступа к тем или иным объектам.

*Рабочей* называется группа пользователей, работающих с од­ной базой данных и имеющих общий файл.

*Файл рабочей группы* – это системный файл с информацией о группе пользователей, работающих с БД коллективного доступа. В файлах рабочих групп хранятся учетные записи, пароли, а также данные о правах доступа к объектам БД.

Изменение стандартного или создание нового файла рабочей Группы выполняет программа Администратор рабочих групп, находящаяся в папке System папки Windows.

После установки Access пользователь получает право доступа ко всем объектам БД, т. Е. становится членом группы Admins с Именем Admin. Члены группы Admins (администраторы) имеют Право на модификацию БД.

Чтобы устранить произвольный доступ в систему всех членов группы Admins, следует установить пароль для каждого админист­ратора в ее регистрационной записи. В противном случае при каждом запуске Access администратор будет регистрироваться как пользователь Admin, т. Е. без указания пароля.

После создания рабочей группы можно приступить К созданию учетных (регистрационных) записей. По умолчанию создается учетная запись Admin, а также учетные записи групп Admins (Адми­нистраторы) и Users (Пользователи) и предоставляются права доступа ко всем объектам.

Учетная запись администратора включается в рабочую ГPYIlI1I Admins. Администратор имеет право доступа ко всем объекта созданным в этой группе.

Кроме администратора в рабочей группе может быть указа владелец базы данных (в системе обеспечения безопасности Аcсеss владельцы объектов имеют особый статус). По умолчанию пользователь, создавший объект, становится владельцем этого объекта и имеет право на работу с ним.

Администраторы и владельцы имеют следующие права:

* администратор БД всегда может получить право доступа ко всем объектам, созданным членами данной рабочей группы;
* владелец БД всегда может открыть базу данных;
* владелец объекта наделен полными правами доступа к этому объекту.

Пользователь Admin является владельцем любой базы данных и всех объектов. Поскольку для него не устанавливается пароль, то для защиты БД от несанкционированного доступа необходимо изменить право владения базой и ее объектами.

Существует несколько способов смены владельца объектов базы данных:

* импортирование всех объектов БД в новый файл;
* использование вкладки *Смена владельца* диалогового окна *Раз­решения.*

Администратор БД предоставляет всем или некоторым членам рабочей группы права доступа к различным объектам базы дан­ных. Права доступа хранятся в файле базы данных и характеризу­ют ее объекты.

К разграничению прав доступа пользователей и групп присту­пают после создания рабочей группы, определения администра­тора и владельца базы данных, а также создания учетных записей пользователей и групп. Пользователь наследуют права той груп­пы, к которой принадлежит.

Перечень прав доступа, определенных в Access, приведен в таблице:

|  |  |  |
| --- | --- | --- |
| **Право доступа** | **Действие** | **Объект доступа** |
| Открытие\запуск | Открытие БД, формы, отчета или запуск макроса | БД, формы, отчеты макросы |
| Чтение макета | Просмотр объектов в режиме Конструктора | Таблицы, запросы, формы, отчеты, макросы и модули |
| Изменение макета | Просмотр, изменение и удаление объектов в режиме Конструктора | То же |
| Администратора | Полный доступ к объектам и данным, включая возможность присвоения прав доступа | БД, таблицы, запросы, формы, отчеты, макросы и модули |
| Чтение данных | Просмотр данных | То же |
| Обновление данных | Просмотр и изменение данных без вставки и удаления | То же |
| Вставка данных | Просмотр и вставка данных без изменения и удаления | То же |
| Удаление данных | Просмотр и удаление данных | То же |
| Монопольный доступ | Открытие БД в монопольном режиме | Базы данных |

Из таблицы видно, что некоторые права доступа обуславлива­ют наличие других прав. Так, таблица, в которой разрешено об­новление данных, доступна для чтения данных и макета. При на­личии прав администратора пользователю доступны все перечисленные выше права.

Определить права доступа к некоторому объекту может владе­лец этого объекта, администратор в рабочей группе Admins или пользователь, которому присвоены права администратора имен­но для этого объекта.

Все права доступа к объекту сохраняются при его изменении только в том случае, если не применялся буфер обмена или не выполнялся импорт/экспорт объекта. Однако все связанные

Объектом права доступа могут быть утеряны при сохранении объекта под новым именем посредством команды *Сохранить как \ Экспорт*.

Еще одним способом защиты является шифрование – это за­щита БД от несанкционированного доступа с помощью текстового редактора или средств работы с файлами, например, входящих в состав Windows или Norton Utilities. Информация в зашифро­ванной базе данных недоступна для чтения. Шифрование несколько замедляет работу Access, так как на шифрование и дешифрование файлов расходуется время.

Шифрование и дешифрование базы данных могут производить только члены группы Admins.