Частное профессиональное образовательное учреждение
«СЕВЕРО-КАВКАЗСКИЙ КОЛЛЕДЖ ИННОВАЦИОННЫХ ТЕХНОЛОГИЙ»
(ЧПОУ «СККИТ»)

ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

КУРС ЛЕКЦИЙ

09.02.06 Сетевое и системное администрирование

Пятигорск

Содержание

Лекция 1. Основные понятия баз данных

План:

1. Основные понятия теории баз данных
2. История возникновения баз данных
3. История развития баз данных
4. Классификация БД

Компьютеры были созданы для решения вычислительных задач, однако со временем они все чаще стали использоваться для построения систем обработки документов, а точнее, содержащейся в них информации. Такие системы обычно и называют информационными.

Информационные системы имеют следующие особенности:

- для обеспечения их работы нужны сравнительно низкие вычислительные мощности
- данные, которые они используют, имеют сложную структуру
- необходимы средства сохранения данных между последовательными запусками системы, …

Другими словами, информационная система требует создания в памяти ЭВМ *динамически обновляемой* модели внешнего мира с использованием единого хранилища - **базы данных (БД)**.

Предметная область - часть реального мира, подлежащая изучению с целью организации управления и, в конечном счете, автоматизации.

Отличительной чертой баз данных следует считать то, что данные хранятся совместно с их описанием, а в прикладных программах описание данных не содержится. Независимые от программ пользователя данные обычно называются **метаданными**. В ряде современных систем метаданные, содержащие также информацию о пользователях, форматы отображения, статистику обращения к данным и др. сведения, *хранятся в словаре базы данных*.

Таким образом, **система управления базой данных** (*СУБД*) - важнейший компонент информационной системы. Для создания и управления информационной системой СУБД необходима в той же степени, как для разработки программы на алгоритмическом языке необходим транслятор.

Основные функции СУБД:

- управление данными во внешней памяти (на дисках);
- управление данными в оперативной памяти;
- журнализация изменений и восстановление базы данных после сбоев;
- поддержание языков БД (язык определения данных, язык манипулирования данными).

История возникновения БД

В истории вычислительной техники можно проследить развитие двух основных областей ее использования.

Первая область — применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно

производить вручную. Характерной особенностью данной области применения вычислительной техники является наличие сложных алгоритмов обработки, которые применяются к простым по структуре данным, объем которых сравнительно невелик.

Вторая область — это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. Информационная система представляет собой программно-аппаратный комплекс, обеспечивающий выполнение следующих функций:

1. надежное хранение информации в памяти компьютера;
2. выполнение специфических для данного приложения преобразований информации и вычислений;
3. предоставление пользователям удобного и легко осваиваемого интерфейса.

Обычно такие системы имеют дело с большими объемами информации, имеющей достаточно сложную структуру.

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

Пользователи видят файл как линейную последовательность записей и могут выполнить над ним ряд стандартных операций:

- создать файл (требуемого типа и размера);
- открыть ранее созданный файл;
- прочитать из файла некоторую запись (текущую, следующую, предыдущую, первую, последнюю);
- записать в файл на место текущей записи новую, добавить новую запись в конец файла.

В разных файловых системах эти операции могли несколько отличаться, но общий смысл их был именно таким.

Главное, что следует отметить, это то, что **структура записи файла была известна только программе, которая с ним работала**, система управления файлами не знала ее. И поэтому для того, чтобы извлечь некоторую информацию из файла, необходимо было точно знать структуру записи файла с точностью до бита. Каждая программа, работающая с файлом, должна была иметь у себя внутри структуру данных, соответствующую структуре этого файла. Поэтому **при изменении структуры файла требовалось изменять структуру программы**, а это требовало новой компиляции, то есть процесса перевода программы в исполняемые машинные коды. Такая ситуация характеризовалась как зависимость программ от данных. Для информационных систем характерным является наличие большого числа различных пользователей (программ), каждый из которых имеет свои специфические алгоритмы обработки информации, хранящейся в одних и тех же

файлах. **Изменение структуры файла, которое было необходимо для одной программы, требовало исправления и перекомпиляции и дополнительной отладки всех остальных программ, работающих с этим же файлом.** Это было первым существенным недостатком файловых систем, который явился толчком к созданию новых систем хранения и управления информацией.
Поскольку файловые системы являются общим хранилищем файлов, принадлежащих, вообще говоря, разным пользователям, системы управления файлами должны обеспечивать авторизацию доступа к файлам. В общем виде подход состоит в том, что **по отношению к каждому зарегистрированному пользователю данной вычислительной системы для каждого существующего файла указываются действия, которые разрешены или запрещены данному пользователю**. И отсутствие централизованных методов управления доступом к информации послужило еще одной причиной разработки СУБД.
Следующей причиной стала **необходимость обеспечения эффективной параллельной работы многих пользователей с одними и теми же файлами**.
В общем случае системы управления файлами обеспечивали режим многопользовательского доступа. Если операционная система поддерживает многопользовательский режим, вполне реальна ситуация, когда два или более пользователя одновременно пытаются работать с одним и тем же файлом. Если все пользователи собираются только читать файл, ничего страшного не произойдет. Но если хотя бы один из них будет изменять файл, для корректной работы этих пользователей требуется взаимная синхронизация их действий по отношению к файлу.
В системах управления файлами обычно применялся следующий подход. В операции открытия файла (первой и обязательной операции, с которой должен начинаться сеанс работы с файлом) среди прочих параметров указывался режим работы (чтение или изменение). Если к моменту выполнения этой операции некоторым пользовательским процессом PR1 файл был уже открыт другим процессом PR2 в режиме изменения, то в зависимости от особенностей системы процессу PR1 либо сообщалось о невозможности открытия файла, либо он блокировался до тех пор, пока в процессе PR2 не выполнялась операция закрытия файла.
При подобном способе организации одновременная работа нескольких пользователей, связанная с модификацией данных в файле, либо вообще не реализовывалась, либо была очень замедлена.
Эти недостатки послужили тем толчком, который заставил разработчиков информационных систем предложить новый подход к управлению информацией. Этот подход был реализован в рамках новых программных систем, названных впоследствии Системами Управления Базами Данных (СУБД), а сами хранилища информации, которые работали под управлением данных систем, назывались базами или банками данных (БД и БнД).

История развития БД

Концепция БД сложилась в конце 60-х годов прошлого столетия и с тех пор постоянно развивалась.

Первый этап сложился к началу 60-х годов прошлого века и характеризуется следующими признаками:

- информация преимущественно хранится в последовательных файлах на магнитных лентах;
- физическая структура данных строго соответствует логической;
- в качестве архива хранятся несколько копий файлов;
- файлы предназначены для единственной программы;
- программист планирует не только логическую, но и физическую организацию данных;
- при изменении физической или логической организации данных программа должна перерабатываться.

Второй этап относится к середине 60-х годов и имеет следующие особенности:

- появились внешние устройства прямого доступа, позволившие осуществить произвольный доступ к записям (прямой, индексно-последовательный);
- вошли в употребление процедуры поиска записи по ключевому полю (обычно одному);
- стало возможным переносить файлы на другие внешние устройства без изменения прикладных программ, что обычно обеспечивалось средствами языка управления данными соответствующей операционной системы.

Третий этап начался с конца 60-х годов. Основным достижением можно считать осознание необходимости централизации данных для доступа к ним различных приложений. При этом уменьшается избыточность и противоречивость информации, приложения используют стандартные средства доступа к данным. На этом этапе возросла сложность организации данных, был реализован эффективный поиск записей по многим ключам.

Именно на этом этапе появились первые СУБД. Прежде всего развивались теория и практика построения иерархических и сетевых СУБД. В этих моделях связи данных описываются с помощью деревьев и графов общего вида.

Четвертый этап датируется второй половиной 70-х годов. На этом этапе были реализованы следующие основные характеристики СУБД:

- логическая и физическая независимость данных;
- удобство развития БД;
- безопасность, секретность, целостность данных;
- поиск информации по различным запросам;
- языковые средства для администратора, прикладного программиста, пользователя-непрофессионала.

С начала 70-х годов после публикаций Э.Кодда начались активные исследования реляционной модели данных. Основу реляционной СУБД составляют таблицы. Вплоть до 80-х годов реляционные СУБД считались перспективными, но трудными для реализации.

Новый этап в развитии СУБД наступил при появлении персональных компьютеров. На этом этапе на передний план вышли такие особенности СУБД, как:

- дружественность и удобство работы пользователя (развитые диалоги, меню, оконный интерфейс, контекстная помощь);
- упрощение громоздких схем СУБД за счет частичной реализации ряда свойств;

- почти полный переход на реляционные СУБД;
- ориентация не только на программиста, но и на пользователя-непрофессионала;
- наличие средств автоматизации программирования в виде генераторов форм, меню, отчетов, запросов.

Классификация БД

Классификация БД может быть произведена по различным признакам, среди которых выделяют:

1. *По форме представления информации:* фактографические и документальные.
2. *По типу используемой модели данных:* иерархические, сетевые, реляционные.
3. *По типологии хранения данных:* локальные (централизованные) и распределённые (удалённые) БД.
   Классификация не является полной. Различные источники предоставляют разнообразную классификацию.
   Вопросы для самоконтроля:

1. Дайте определения понятиям: информационная система, предметная область.
2. Что называется базой данных и каково ее место в ИС?
3. В чем различие между данными и метаданными?
4. Каково назначение систем управления базами данных?
5. Для чего используется словарь данных?
6. Назовите этапы развития БД.
7. Какую роль в развитии технологии БД сыграло появление ПК?
8. Каковы функции СУБД?

Технологии работы с базами данных

План:

1. Централизованная архитектура
2. Архитектура "файл-сервер"
3. Технология "клиент – сервер"
4. Трехзвенная (многозвенная) архитектура "клиент – сервер"

Понятие базы данных изначально предполагало возможность решения многих задач несколькими пользователями. В связи с этим, важнейшей характеристикой современных СУБД является наличие многопользовательской технологии работы. Разная реализация таких технологий в разное время была связана как с основными свойствами вычислительной техники, так и с развитием программного обеспечения.

Централизованная архитектура

При использовании этой технологии база данных, СУБД и прикладная программа (приложение) располагаются на одном компьютере (рисунок 1). Для такого способа организации не требуется поддержки сети и все сводится к автономной работе.

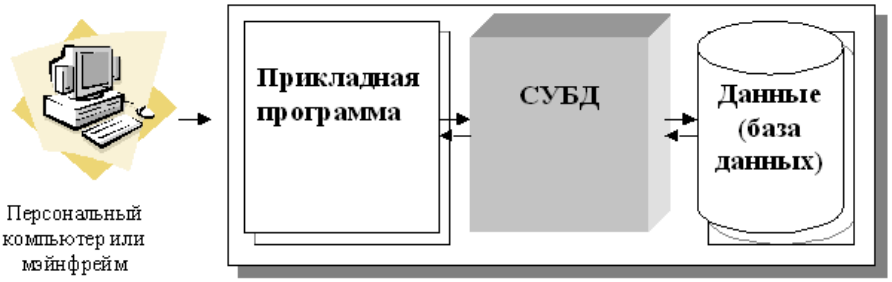

Рисунок 1 - Централизованная архитектура

Многопользовательская технология работы обеспечивалась либо режимом мультипрограммирования, либо режимом разделения времени.

Архитектура "файл-сервер"

Увеличение сложности задач, появление персональных компьютеров и локальных вычислительных сетей явились предпосылками появления новой архитектуры *файл-сервер*. Эта архитектура баз данных с сетевым доступом предполагает назначение одного из компьютеров сети в качестве выделенного сервера, на котором будут храниться файлы базы данных. В соответствии с запросами пользователей файлы с *файл-сервера* передаются на рабочие станции пользователей, где и осуществляется основная часть обработки данных. Центральный сервер выполняет в основном только роль хранилища файлов, не участвуя в обработке самих данных (рисунок 2).

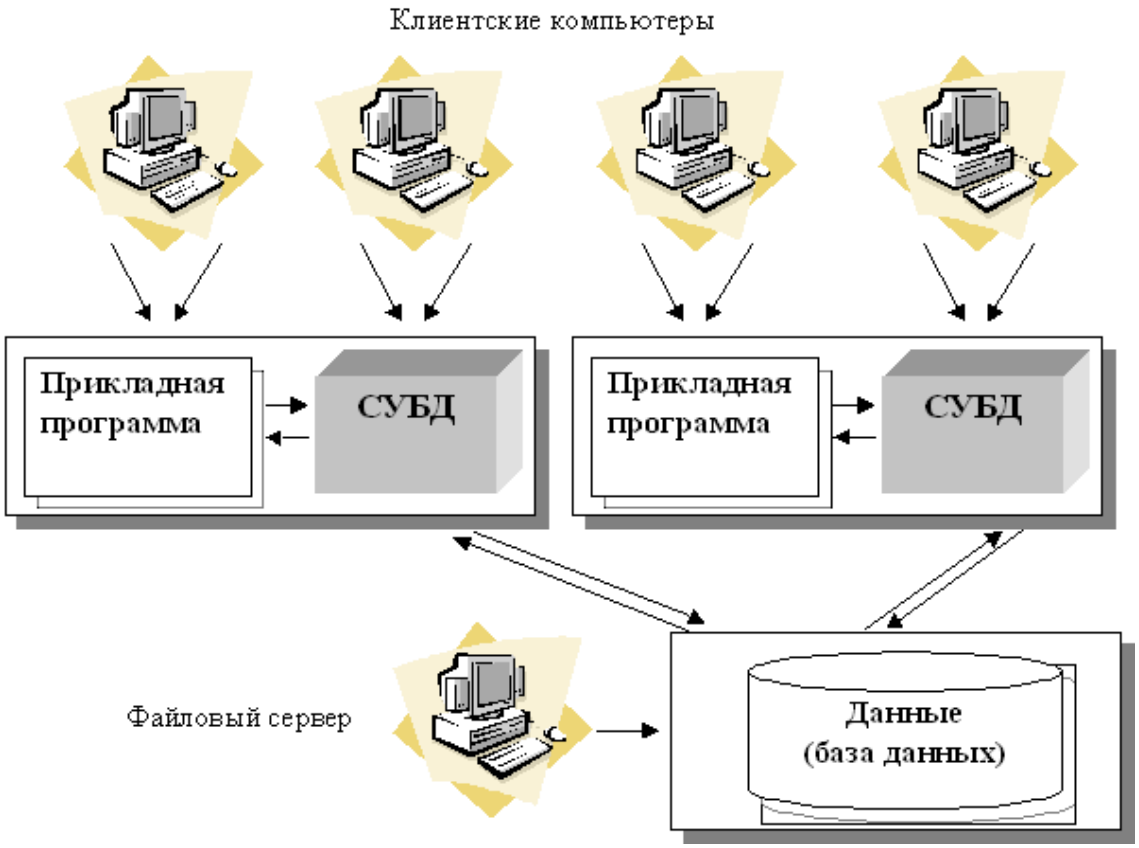

Рисунок 2 - Архитектура "файл-сервер"

Технология "клиент – сервер"

Использование технологии " *клиент – сервер* " предполагает наличие некоторого количества компьютеров, объединенных в сеть, один из которых выполняет особые управляющие функции (является сервером сети).
Так, архитектура " *клиент – сервер* " разделяет функции приложения пользователя (называемого клиентом) и сервера. Приложение-клиент формирует запрос к серверу, на котором расположена БД, на структурном языке запросов SQL (Structured Query Language), являющемся промышленным стандартом в мире реляционных БД. Удаленный сервер принимает запрос и переадресует его SQL-серверу БД.
SQL-сервер – специальная программа, управляющая удаленной базой данных. SQL-сервер обеспечивает интерпретацию запроса, его выполнение в базе данных, формирование результата выполнения запроса и выдачу его приложению-клиенту. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса; клиентский компьютер лишь отсылает запрос к серверной БД и получает результат, после чего интерпретирует его необходимым образом и представляет пользователю. Так как клиентскому приложению посылается результат выполнения запроса, по сети "путешествуют" только те данные, которые необходимы клиенту. В итоге снижается нагрузка на сеть. Поскольку выполнение запроса происходит там же, где хранятся данные (на сервере), нет необходимости в пересылке больших

пакетов данных. Кроме того, SQL-сервер, если это возможно, оптимизирует полученный запрос таким образом, чтобы он был выполнен в минимальное время с наименьшими накладными расходами.
Архитектура системы представлена на рисунке 3.

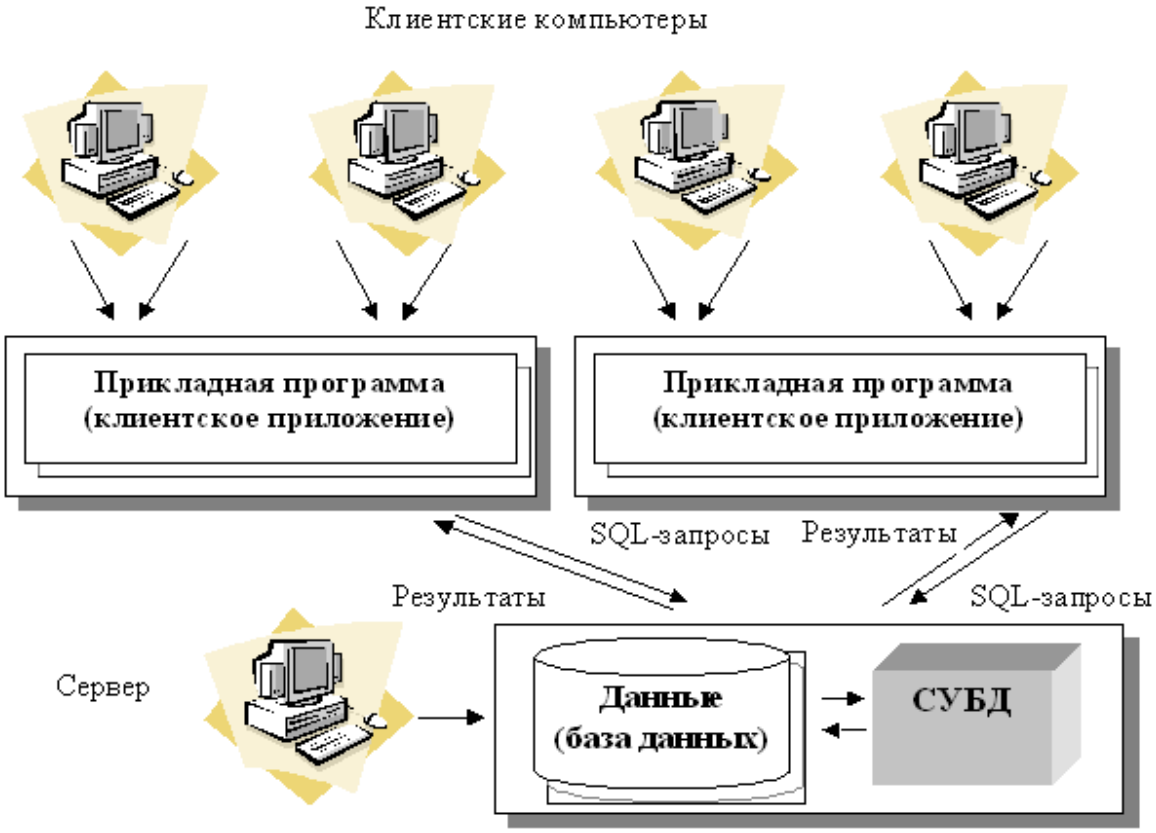

Рисунок 3 - Архитектура "клиент – сервер"

Трехзвенная (многозвенная) архитектура "клиент – сервер"

Трехзвенная (в некоторых случаях *многозвенная*) **архитектура** (N-tier или multi-tier). представляет собой дальнейшее совершенствование технологии " *клиент – сервер* ". Рассмотрев архитектуру " *клиент – сервер* ", можно заключить, что она является 2-звенной: первое звено – клиентское приложение, второе звено – сервер БД + сама БД. В *трехзвенной архитектуре* вся бизнес-логика (деловая логика), ранее входившая в клиентские приложения, выделяется в отдельное звено, называемое сервером приложений. При этом клиентским приложениям остается лишь пользовательский интерфейс. Схематически такую *архитектуру* можно представить, как показано на рисунке 4.

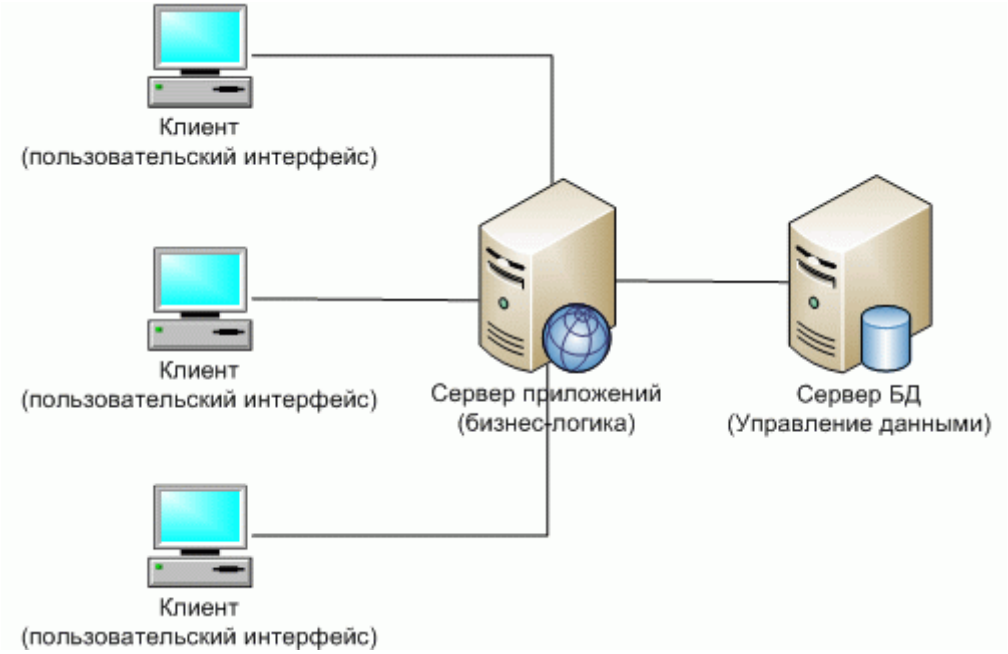

Рисунок 4 - Представление многоуровневой архитектуры "клиент-сервер"

Вопросы для самоконтроля:

1. Назовите достоинства и недостатки существующих многопользовательских технологий с базами данных.

Логическая и физическая независимость данных

План:

1. Базовые понятия
2. Архитектура базы данных
3. Механизм прохождения запроса к БД

Современные авторы часто употребляют термины "банк данных" и "база данных" как синонимы, однако в общеотраслевых руководящих материалах по созданию банков данных Государственного комитета по науке и технике (ГКНТ), изданных в 1982 г., эти понятия различаются. Там приводятся следующие определения банка данных, базы данных и СУБД:
Банк данных (БнД) — это система специальным образом организованных данных — баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.
База данных (БД) — именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области.

Система управления базами данных (СУБД) — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.
Программы, с помощью которых пользователи работают с базой данных, называются *приложениями*. В общем случае с одной базой данных могут работать множество различных приложений. При рассмотрении приложений, работающих с одной базой данных, предполагается, что они могут работать параллельно и независимо друг от друга, и именно СУБД призвана обеспечить работу множества приложений с единой базой данных таким образом, чтобы каждое из них выполнялось корректно, но учитывало все изменения в базе данных, вносимые другими приложениями.

Архитектура базы данных

В процессе научных исследований, посвященных тому, как именно должна быть устроена СУБД, предлагались различные способы реализации.
Самым жизнеспособным из них оказалась предложенная американским комитетом по стандартизации ANSI (American National Standards Institute) трехуровневая система организации БД, изображенная на рисунке 5.

Рисунок 5 - Трехуровневая модель системы управления базой данных, предложенная ANSI

Уровень внешних моделей — самый верхний уровень, где каждая модель имеет свое "видение" данных. Этот уровень определяет точку зрения на БД отдельных приложений. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению.
Концептуальный уровень — центральное управляющее звено, здесь база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную модель

предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.
Физический уровень — собственно данные, расположенные в файлах или в страничных структурах, расположенных на внешних носителях информации.
Эта архитектура позволяет обеспечить логическую (между уровнями 1 и 2) и физическую (между уровнями 2 и 3) независимость при работе с данными.
Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же базой данных.
Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с данной базой данных.

Процесс прохождения пользовательского запроса

Рисунок 6 иллюстрирует взаимодействие пользователя, СУБД и ОС при обработке запроса на получение данных. Цифрами помечена последовательность взаимодействий:

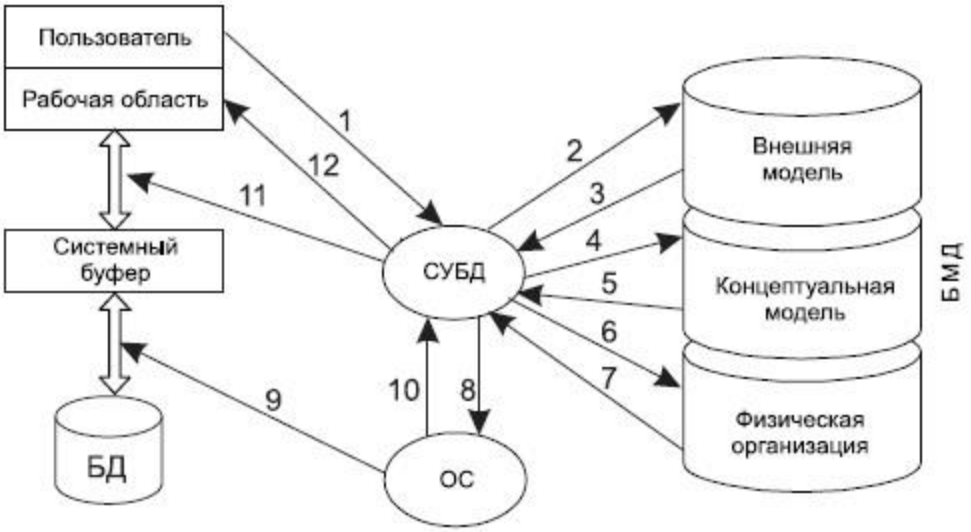

Рисунок 6 - Схема прохождения запроса к БД

Разумеется, механизм прохождения запроса в реальных СУБД гораздо сложнее, но и эта упрощенная схема показывает, насколько серьезными и сложными должны быть механизмы обработки запросов, поддерживаемые реальными СУБД.

Вопросы для самоконтроля:

1. Каким образом прикладные программы взаимодействуют с БД?
2. Чем банк данных отличается от базы данных?
3. Какие компоненты входят в состав банка данных?
4. Что представляет собой трехуровневая архитектура СУБД?

5. В чем особенность уровня внешних моделей?
6. В чем особенность концептуального уровня?
7. В чем особенность физического уровня?
8. Что означает логическая и физическая независимость данных?

Лекция 2. Взаимосвязи в моделях и реляционный подход к построению моделей

План:

1. Иерархическая модель базы данных
2. Сетевая модель базы данных
3. Реляционная модель базы данных

Различают три основные модели базы данных - это иерархическая, сетевая и реляционная. Эти модели отличаются между собой по способу установления связей между данными.

Иерархическая модель базы данных

Иерархические базы данных - самая ранняя модель представления сложной структуры данных. Информация в иерархической базе организована по принципу древовидной структуры, в виде отношений "предок-потомок".
Каждая запись может иметь не более одной родительской записи и несколько подчиненных.
Связи записей реализуются в виде физических указателей с одной записи на другую.
Основной недостаток иерархической структуры базы данных - невозможность реализовать отношения "многие-ко-многим", а также ситуации, когда запись имеет несколько предков.
Иерархические базы данных графически могут быть представлены как перевернутое дерево, состоящее из объектов различных уровней. Верхний уровень (корень дерева) занимает один объект, второй - объекты второго уровня и так далее.

Организация данных в СУБД иерархического типа определяется в терминах: элемент, агрегат, запись (группа), групповое отношение, база данных.

- **Атрибут** (элемент данных) - наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Элемент данных также часто называют полем.

- **Запись** - именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Тип записи определяется составом ее атрибутов. Экземпляр записи - конкретная запись с конкретным значением элементов.
- **Групповое отношение** - иерархическое отношение между записями двух типов. Родительская запись (владелец группового отношения) называется исходной записью, а дочерние записи (члены группового отношения) - подчиненными. Иерархическая база данных может хранить только такие древовидные структуры.

  Корневая запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках группового отношения. Каждая запись идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой, по иерархическому пути.
  Для групповых отношений в иерархической модели обеспечивается автоматический режим включения и фиксированное членство. Это означает, что для запоминания любой некорневой записи в БД должна существовать ее родительская запись.

  Операции над данными, определенные в иерархической модели:
- **Добавить** в базу данных новую запись. Для корневой записи обязательно формирование значения ключа.
- **Изменить** значение данных предварительно извлеченной записи. Ключевые данные не должны подвергаться изменениям.
- **Удалить** некоторую запись и все подчиненные ей записи.
- **Извлечь** корневую запись по ключевому значению, допускается также последовательный просмотр корневых записей.
- **Извлечь** следующую запись (следующая запись извлекается в порядке левостороннего обхода дерева).
  В операции **ИЗВЛЕЧЬ** допускается задание условий выборки.
  Все операции изменения применяются только к одной "текущей" записи (которая предварительно извлечена из базы данных). Такой подход к манипулированию данных получил название "навигационного".
  В иерархической БД поддерживается только целостность связей между владельцами и членами группового отношения (никакой потомок не может существовать без предка).

Сетевая модель базы данных

Сетевая модель данных определяется в тех же терминах, что и *иерархическая*. Она состоит из множества записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также имеет вид 1:N.
Основное различие этих моделей состоит в том, что в сетевой модели запись может быть членом более чем одного группового отношения.

Операции над данными в сетевой модели БД

- **Добавить** - внести запись в БД и, в зависимости от режима включения, либо включить ее в групповое отношение, где она объявлена подчиненной, либо не включать ни в какое групповое отношение.
- **Включить в групповое отношение** - связать существующую подчиненную запись с записью-владельцем.
- **Переключить** - связать существующую подчиненную запись с другой записью-владельцем в том же групповом отношении.
- **Обновить** - изменить значение элементов предварительно извлеченной записи.
- **Извлечь** - извлечь записи последовательно по значению ключа, а также используя групповые отношения - от владельца можно перейти к записям - членам, а от подчиненной записи к владельцу набора.
- **Удалить** - убрать из БД запись. Если эта запись является владельцем группового отношения, то анализируется класс членства подчиненных записей. Обязательные члены должны быть предварительно исключены из группового отношения, фиксированные удалены вместе с владельцем, необязательные останутся в БД.
- **Исключить из группового отношения** - разорвать связь между записью-владельцем и записью-членом.

В сетевой модели обеспечивается только поддержание целостности по ссылкам (владелец отношения - член отношения).

Реляционная модель базы данных

Реляционная модель была предложена в 1969 году сотрудником фирмы IBM Е. Ф. Коддом (Or. E. P. Codd), известным исследователем в области баз данных. Впервые основные концепции этой модели были опубликованы в 1970 году. Набор средств для управления реляционными базами данных называется реляционной системой управления базами данных (РСУБД). Реляционная система управления базами данных может содержать утилиты, приложения, сервисы, библиотеки, средства создания приложений и другие компоненты. Реляционная база данных представляет собой совокупность двумерных таблиц. Любая таблица реляционной базы данных состоит из строк, называемых *записями,* и столбцов, называемых *полями.* Строки таблицы содержат сведения об объектах. Каждый столбец в таблице должен содержать только определенный тип информации. Каждая *строка* таблицы содержит разнообразную (разного типа) информацию. Все данные, помещенные в одной строке, называют *записью,* каждый *элемент записи* - это *поле.* Таким образом, каждое поле содержит часть информации, находящейся на пересечении соответствующей строки и столбца. В таблице всевозможные значения одного типа в одном столбце называют *доменом.*
Поле является элементом записи и представляет собой ячейку таблицы. У каждого столбца есть свое неповторимое имя, описывающее тот вид информации, который содержится в нем. Это имя называют *«именем поля* базы данных»

Каждое поле имеет фиксированную длину, следовательно, и любая запись в таблице имеет фиксированную длину. Каждая запись характеризуется своим уникальным порядковым номером.
Данные в реляционной таблице должны удовлетворять следующим принципам:
1. Каждое значение поля должно быть атомарным, т.е. не расчленяемым на несколько значений;
2. Значения данных домена (в одном и том же столбце) должны принадлежать к одному и тому же типу данных, доступному для использования в данной СУБД;
3. Каждая запись в таблице уникальна, т.е. в таблице не существует двух записей с полностью совпадающим набором значений ее полей;
4. Каждое поле имеет уникальное имя;
5. Последовательность полей в таблице несущественна;
6. Последовательность записей в таблице несущественна.
Существенное отличие реляционной модели от обыкновенного последовательного файла заключается в том, что все столбцы в таблице с точки зрения входа предполагаются эквивалентными. Именно это свойство делает эту модель весьма мощной и делает невозможным отображение ее на память в виде последовательного массива данных.
Поскольку записи в таблице неупорядочены, то необходимо указать поле (или набор нескольких полей) для уникальной идентификации каждой записи.
Первичный ключ - это поле или набор полей, которые однозначно идентифицируют (определяют) запись таблицы
Обычно ключом является поле или совокупность полей фиксированной длины. Каждому значению *первичного* ключа соответствует одна и только одна запись. Первичный ключ любой таблицы обязан содержать уникальные непустые значения для каждой записи. Если первичный ключ состоит из нескольких полей, он называется *составным первичным ключом* (primary key).
Поле, указывающее на запись в другой таблице, связанную с данной записью, называется *внешним ключом* (foreign key).
Подобное взаимоотношение между таблицами называется *связью* (relationship). Связь между двумя таблицами устанавливается путем присвоения значений внешнего ключа одной таблицы значениям первичного ключа другой. Таблица, содержащая внешний ключ, называется *второстепенной*, а таблица, содержащая первичный ключ, определяющий возможные значения внешнего ключа второстепенной таблицы, называется *главной*.
Типичная реляционная база данных состоит из нескольких связанных таблиц.

Типы связей между объектами

Все информационные объекты предметной области связаны между собой. Соответствия, отношения, возникающие между объектами предметной области, называются *связями.* Связанные отношениями таблицы взаимодействуют по принципу *главная*, *подчиненная*. Возможны следующие отношения между таблицами:
1. Отношение *«один – ко – многим»* (обозначают 1:M): одной записи из главной таблицы может соответствовать ноль, одна или несколько записей подчинённой таблицы.

2. ***Отношение «один – к - одному»*** (обозначают 1:1): одной записи из главной таблицы соответствует только одна запись из подчинённой таблицы.
3. ***Отношение «многие – ко – многим»*** (обозначают 1:1): одной записи из главной таблицы может соответствовать ноль, одна или несколько записей подчинённой таблицы и наоборот.

Одним из правил ссылочной целостности (referential integrity) является то, что первичный ключ любой таблицы должен содержать уникальные непустые значения для данной таблицы. Некоторые СУБД могут контролировать уникальность первичных ключей. Если СУБД контролирует уникальность первичных ключей, то при попытке присвоить первичному ключу значение, уже имеющееся в другой записи, СУБД сгенерирует диагностическое сообщение, обычно содержащее словосочетания primary key violation. Это сообщение в дальнейшем может быть передано в приложение, с помощью которого конечный пользователь манипулирует данными.
Если две таблицы связаны соотношением главная-подчиненная, внешний ключ подчинённой таблицы должен содержать только те значения, которые имеются среди значений первичного ключа главной таблицы. Если корректность значений внешних ключей не контролируется СУБД, можно говорить о нарушении ссылочной целостности. Если же СУБД контролирует корректность значений внешних ключей, то при попытке присвоить внешнему ключу значение, отсутствующее среди значений первичных ключей главной таблицы, либо при удалении или модификации записей главной таблицы, приводящих к нарушению ссылочной целостности, СУБД сгенерирует сообщение, о котором говорилось выше.

Вопросы для самоконтроля:

1. Что такое модель данных?
2. Для чего строится модель данных?
3. Укажите достоинства и недостатки иерархической модели данных.
4. Как организуется физическое размещение данных в БД иерархического типа?
5. Охарактеризуйте сетевую модель данных.
6. Охарактеризуйте реляционную модель данных.
7. Чем отличается реляционная модель данных от предшествующих ей моделей?
8. Что такое простой ключ и составной ключ?
9. Перечислите виды связей между объектами? Охарактеризуйте их.
10. Как проявляется иерархическая подчиненность в связи «один ко многим»?

Реляционная алгебра

План:

1. Традиционные операции реляционной алгебры
2. Специальные операции реляционной алгебры

С точки зрения внешнего представления объектов реального мира *модель данных* — это основные понятия и способы, используемые при анализе и описании предметной области.
Среди многих попыток представить обработку данных на формальном абстрактном уровне реляционная модель, предложенная Э. Ф. Коддом, стала по существу первой работоспособной *моделью данных,* поскольку помимо средств описания объектов имела эффективный инструментарий преобразований этих описаний — операции реляционной алгебры.
Реляционная алгебра в том виде, в котором она была определена Э. Ф. Коддом, состоит из двух групп по четыре оператора.
1. Традиционные операции: объединение, пересечение, разность и декартово произведение.
2. Специальные реляционные операции: выборка, проекция, соединение, деление.
Объединение возвращает таблицу, содержащую все записи, которые принадлежат либо одной из двух заданных таблиц, либо им обоим.
Пересечение возвращает таблицу, содержащую все записи, которые принадлежат одновременно двум заданным таблицам.
Разность возвращает таблицу, содержащую все записи, которые принадлежат первому из двух заданных таблиц и не принадлежат второй.
Произведение возвращает таблицу, содержащую все возможные записи, которые являются сочетанием двух записей, принадлежащих соответственно двум заданным таблицам.
Выборка возвращает таблицу, содержащую все записи из заданной таблицы, которые удовлетворяют указанным условиям.
Проекция возвращает таблицу, содержащую все записи заданной таблицы, которые остались в этой таблицы после исключения из неё некоторых атрибутов
Соединение возвращает таблицу, содержащую все возможные записи, которые представляют собой комбинацию атрибутов двух записей, принадлежащих двум заданным таблицам, при условии, что в этих двух комбинированных записях присутствуют одинаковые значения в одном или нескольких общих для исходных таблиц атрибутах (причем эти общие значения в результирующем записи появляются один раз, а не дважды).
Деление для заданных двух унарных таблиц и одной бинарной возвращает таблицу, содержащую все записи из первой унарной таблицы, которые содержатся также в бинарной таблице и соответствуют всем записям во второй унарной таблице.

Результат выполнения любой операции над таблицами также является таблицей, поэтому результат одной операции может использоваться в качестве исходных данных для другой. Другими словами, можно записывать вложенные реляционные выражения, т. е. выражения, в которых операторы сами представлены реляционными выражениями, причем произвольной сложности. Эта особенность называется свойством *реляционной замкнутости.*

Вопросы для самоконтроля:

1. Сколько реляционных операций образуют реляционную алгебру?
2. Перечислите и охарактеризуйте операции реляционной алгебры. Приведите примеры.

Лекция 3. Основные этапы проектирования БД

План:

1. Жизненный цикл БД
2. Планирование разработки базы данных
3. Определение требований к системе
4. Сбор и анализ требований пользователей
5. Проектирование базы данных
6. Разработка приложений
7. Реализация
8. Загрузка данных
9. Тестирование
10. Эксплуатация и сопровождение

Жизненный цикл БД

Как и любой программный продукт, база данных обладает собственным жизненным циклом (ЖЦБД). Главной составляющей в жизненном цикле **БД** является создание единой базы данных и программ, необходимых для ее работы.

ЖЦБД включает в себя следующие основные этапы (рисунок 7):

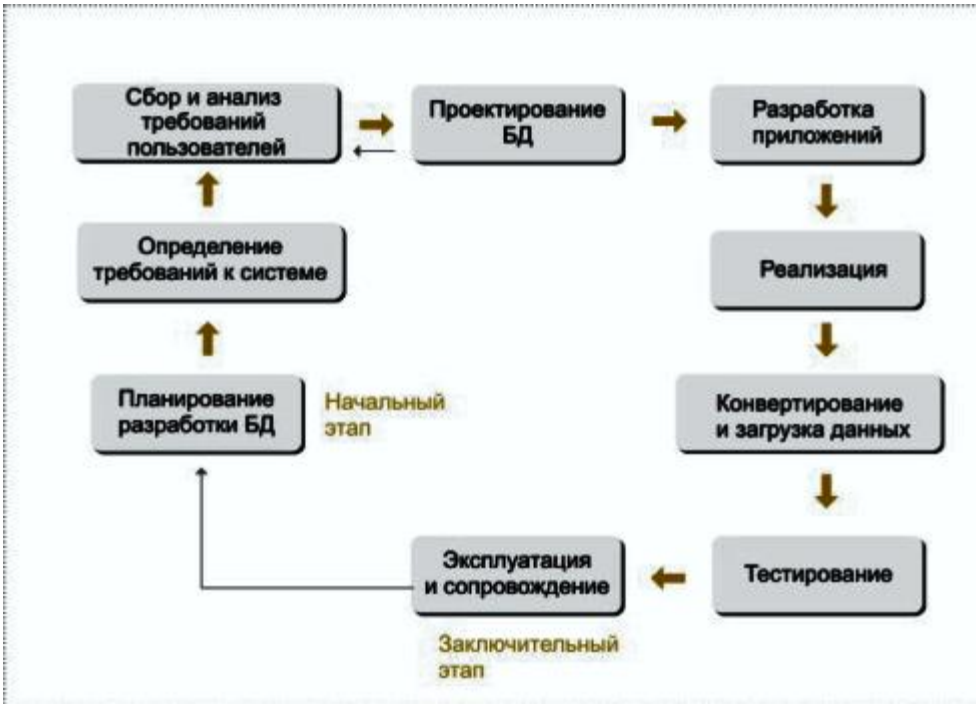

Рисунок 7 - Жизненный цикл БД

Планирование разработки базы данных

Содержание данного этапа — разработка стратегического плана, в процессе которой осуществляется предварительное планирование конкретной системы управления базами данных.
Планирование разработки базы данных состоит в определении трех основных компонентов: объема работ, ресурсов и стоимости проекта.
Важной частью разработки стратегического плана является проверка осуществимости проекта, состоящая из нескольких частей.
Первая часть — проверка технологической осуществимости. Она состоит в выяснении вопроса, существует ли оборудование и программное обеспечение, удовлетворяющее информационным потребностям фирмы.
Вторая часть — проверка операционной осуществимости — выяснение наличия экспертов и персонала, необходимых для работы БД.
Третья часть — проверка экономической целесообразности осуществления проекта. При исследовании этой проблемы весьма важно дать оценку ряду факторов, в том числе и таким:

- целесообразность совместного использования данных разными
- отделами;
- величина риска, связанного с реализацией системы базы данных;
- ожидаемая выгода от внедрения подлежащих созданию приложений;
- время окупаемости внедренной БД;
- влияние системы управления БД на реализацию долговременных
- планов организации.

Определение требований к системе

На данном этапе необходимо определить диапазон действия приложения базы данных, состав его пользователей и области применения.
Определение требований включает выбор целей БД, выяснение информационных потребностей различных отделов и руководителей фирмы и требований к оборудованию и программному обеспечению.

Сбор и анализ требований пользователей

На данном этапе необходимо создать для себя модель движения важных материальных объектов и уяснить процесс документооборота. По каждому документу необходимо установить периодичность использования, определить данные, необходимые для выполнения выделенных функций (анализируя существующую и планируемую документацию, выясняют, как получается каждый элемент данных, кем получается, где в дальнейшем используется, кем контролируется.
Собранная информация о каждой важной области применения приложения и пользовательской группе должна включать следующие компоненты: исходную и генерируемую документацию, подробные сведения о выполняемых транзакциях, а также список требований с указанием их приоритетов.
Формализация собранной на этом этапе информации может быть повышена с помощью методов составления спецификаций требований, к числу которых относятся, например, технология структурного анализа и проектирования, диаграммы потоков данных и графики "вход — процесс — выход".

Проектирование базы данных

Полный цикл разработки базы данных включает концептуальное, логическое и физическое ее проектирование.

Концептуальное проектирование базы данных

Первая фаза процесса проектирования базы данных заключается в создании для анализируемой части предприятия *концептуальной модели данных.*
В построении *общей концептуальной модели данных* выделяют ряд этапов.

- Выделение локальных представлений, соответствующих обычно относительно независимым данным. Каждое такое представление проектируется как подзадача.
- Формулирование сущностей, описывающих локальную предметную область проектируемой БД, и описание атрибутов, составляющих структуру каждой сущности.
- Выделение ключевых атрибутов.
- Спецификация связей между сущностями. Удаление избыточных связей.
- Анализ и добавление неключевых атрибутов.
- Объединение локальных представлений.

Созданная концептуальная модель данных предприятия является источником информации для фазы логического проектирования базы данных.

Логическое проектирование базы данных

Цель второй фазы проектирования базы данных состоит в создании логической модели данных для исследуемой части предприятия.

Логическая модель, отражающая особенности представления о функционировании предприятия одновременно многих типов пользователей, называется *глобальной логической моделью данных.*

Процесс проектирования БД должен опираться на определенную модель данных (реляционная, сетевая, иерархическая), которая определяется типом предполагаемой для реализации информационной системы СУБД.

Концептуальное и логическое проектирование — это итеративные процессы, которые включают в себя ряд уточнений, продолжающиеся до тех пор, пока не будет получен наиболее соответствующий структуре предприятия продукт.

Физическое проектирование базы данных

Целью проектирования на данном этапе является создание описания СУБД ориентированной модели БД.

Действия, выполняемые на этом этапе, слишком специфичны для различных моделей данных, поэтому их сложно обобщить. Остановимся на реляционной модели данных. В этом случае под физическим проектированием подразумевается:

- создание описания набора реляционных таблиц и ограничений для них на основе информации, представленной в глобальной логической модели данных;
- определение конкретных структур хранения данных и методов доступа к ним, обеспечивающих оптимальную производительность системы с базой данных;
- разработка средств зашиты создаваемой системы.

Разработка приложений

Параллельно с проектированием системы базы данных выполняется разработка приложений. Главные составляющие данного процесса — это проектирование транзакций и пользовательского интерфейса.

Проектирование транзакций

Транзакции представляют некоторые события реального мира.

Транзакция может состоять из нескольких операций, однако с точки зрения пользователя эти операции представляют собой единое целое, переводящее базу данных из одного непротиворечивого состояния в другое. Реализация транзакций опирается на тот факт, что СУБД способна обеспечивать сохранность внесенных во время транзакции изменений в БД и непротиворечивость базы данных даже в случае возникновения сбоя.

Проектирование транзакций заключается в определении:

- данных, которые используются транзакцией;
- функциональных характеристик транзакции;
- выходных данных, формируемых транзакцией;
- степени важности и интенсивности использования транзакции.

Проектирование пользовательского интерфейса

Интерфейс должен быть удобным и обеспечивать все функциональные возможности, предусмотренные в спецификациях требований пользователей.

Специалисты рекомендуют при проектировании пользовательского интерфейса использовать следующие основные элементы и их характеристики:

- содержательное название;
- ясные и понятные инструкции;

- логически обоснованные группировки и последовательности полей;
- визуально привлекательный вид окна формы или поля отчета;
- легко узнаваемые названия полей;
- согласованную терминологию и сокращения;
- согласованное использование цветов;
- визуальное выделение пространства и границ полей ввода данных;
- удобные средства перемещения курсора;
- средства исправления отдельных ошибочных символов и целых
- полей;
- средства вывода сообщений об ошибках при вводе недопустимых
- значений;
- особое выделение необязательных для ввода полей;
- средства вывода пояснительных сообщений с описанием полей;
- средства вывода сообщения об окончании заполнения формы.

Реализация

На данном этапе осуществляется физическая реализация базы данных и разработанных приложений, позволяющих пользователю формулировать требуемые запросы к БД и манипулировать данными в БД.
База данных описывается на языке определения данных выбранной СУБД. В результате компиляции его команд и их выполнения создаются схемы и пустые файлы базы данных. На этом же этапе определяются и все специфические пользовательские представления.
Прикладные программы реализуются с помощью языков третьего или четвертого поколений. Кроме того, на этом этапе создаются другие компоненты проекта приложения — например, экраны меню, формы ввода данных и отчеты. Реализация этого, а также и более ранних этапов проектирования БД может осуществляться с помощью инструментов автоматизированного проектирования и создания программ, которые принято называть CASE-инструментами (Computer-Aided Software Engineering).

Загрузка данных

На этом этапе созданные в соответствии со схемой базы данных пустые файлы, предназначенные для хранения информации, должны быть заполнены данными. Наполнение базы данных может протекать по-разному, в зависимости от того, создается ли база данных вновь или новая база данных предназначена для замены старой.

Тестирование

Для оценки законченности и корректности выполнения приложения базы данных может использоваться несколько различных стратегий тестирования:
- нисходящее тестирование;
- восходящее тестирование;
- тестирование потоков;

- интенсивное тестирование.

Эксплуатация и сопровождение

Основные действия, связанные с этим этапом сводятся к наблюдению за созданной системой и поддержке ее нормального функционирования по окончании развертывания.
Поддержка БД предполагает разрешение проблем, возникающих в процессе эксплуатации БД и связанных как с ошибками реализации БД, так и с изменениями в самой предметной области, созданием дополнительных программных компонент или модернизацией самой БД.

Вопросы для самоконтроля:

1. Перечислите этапы, составляющие жизненный цикл БД.
2. Что является целью каждого этапа?
3. Какие работы ведутся на каждом из этапов?

Концептуальное проектирование БД

План:

1. Модель "Сущность - Связь"(ERD)
2. Структурный подход при разработке инфологической модели
3. Моделирование локальных представлений
4. Правила преобразования ER-диаграмм в реляционные таблицы

Концептуальная модель отображает реальный мир в некоторые понятные человеку концепции, полностью независимые от параметров среды хранения данных. Существует множество подходов к построению таких моделей: графовые модели, семантические сети, модель "сущность-связь" и т.д. Наиболее популярной из них оказалась модель "сущность-связь".

Модель "Сущность - Связь"(ERD)

Это модель предметной области, которая используется на этапе концептуального проектирования.
Модель использует три основных элемента: сущность, атрибут и связь.
Сущность - это абстракция какого-либо объекта, процесса или явления реального мира, о котором нужно хранить информацию. В качестве сущности могут выступать материальные (предприятие, товар) и нематериальные (описание явления, реферат статьи) объекты.
Тип сущности определяет набор однородных объектов, а **экземпляр сущности** - конкретный объект в наборе.
Каждый тип сущности обладает одним или несколькими атрибутами.
Каждому типу сущности должно быть дано уникальное имя. К одному и тому же имени должна всегда применяться одна и та же интерпретация.

Для идентификации конкретных экземпляров сущностей используются атрибуты – идентификаторы (один или несколько), которые позволяют однозначно отличать один экземпляр сущности от другого.
Каждая сущность может обладать любым количеством связей с другими сущностями модели.
Атрибут - это поименованная характеристика сущности, которая принимает значения из некоторого множества значений.
Чтобы задать атрибут в модели, необходимо присвоить ему наименование, привести смысловое описание атрибута, определить множество его допустимых значений и указать, для чего он используется.
Основное назначение атрибута - описание свойства сущности, а также идентификация экземпляров сущности.
Атрибут может быть либо обязательным, либо необязательным. Обязательность означает, что атрибут не может принимать неопределенных значений (null values). Атрибут может быть либо описательным, либо входить в состав уникального идентификатора (первичного ключа).
Первичный ключ – набор атрибутов, значения которого однозначно определяют экземпляр сущности.
Внешний ключ - это набор атрибутов, используемый для представления связей между сущностями.
Связь - поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Связь – это средство, с помощью которого представляются отношения между сущностями, имеющие место в предметной области. Связи может даваться имя, выражаемое грамматическим оборотом глагола. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными.
Связи могут быть между двумя (бинарные), тремя (тернарные) и более сущностями. Чаще всего используются бинарные. Они классифицируются следующим образом:
Связь "один - к - одному" (1:1)
Когда каждому экземпляру сущности А соответствует один и только один экземпляр сущности Б, и наоборот. Связь двунаправленная.
Связь "один - ко - многим" (1:M)
Это такой тип связи, когда каждому экземпляру сущности А может соответствовать ни одного, один или несколько экземпляров сущности Б, однако каждому экземпляру сущности Б соответствует один и только один экземпляр сущности А.
Связь "многие - к - одному" (M:1)
Это отображение обратно предыдущему.
Связь "многие - ко - многим" (отображение M:N)
Это такой тип связи, при котором каждому экземпляру сущности А может соответствовать ни одного, один или несколько экземпляров сущности Б, и наоборот.

Информацию о проекте оформляют составлением спецификаций по сущностям, атрибутам и отношениям с использованием графических диаграмм. На диаграмме обозначают:

- сущности - прямоугольниками;
- атрибуты - овалами, соединяя их с соответствующими сущностями ненаправленными ребрами; идентифицирующие атрибуты подчеркиваются;
- связи (отношения) - ромбами, соединяя их с соответствующими сущностями ненаправленными ребрами, за исключением бинарных связей, которые соединяются направленными ребрами.

При моделировании используются следующие общие правила:

- используются только три типа конструктивных элементов - сущность, атрибут и связь;
- в отдельном проектном представлении каждый компонент информации моделируется только одним конструктивным элементом, то есть необходимо избегать избыточности.

Структурный подход при разработке инфологической модели

Сущность структурного подхода к разработке ИС заключается в ее декомпозиции (разбиении) на автоматизируемые функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы "снизу-вверх" от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов. При моделировании предметной области проектировщик разбивает ее на ряд локальных областей, моделирует каждое локальное представление, а затем их объединяет.

Моделирование локальных представлений

Выбор локального представления зависит от масштабов предметной области. Для удобства проектирования желательно использовать в отдельных локальных представлениях шесть - семь сущностей.

1. Формулирование сущностей

На этом этапе необходимо указать типы объектов, о которых нужно хранить информацию. Иногда это сложно сделать, так как отдельный объект можно представить и в виде сущности, и в виде атрибута и в виде связи. Тогда следует проработать несколько вариантов моделей и выбрать наиболее гибкий. Каждой выбранной сущности должно быть присвоено четкое наименование. Общее их количество не должно быть большим.

2. Выбор идентифицирующего атрибута для каждой сущности.

Необходимо для каждой сущности указать идентификатор, позволяющий однозначно распознавать экземпляры сущности. Это один или несколько атрибутов, называемых ключом. Если в наборе атрибутов такого нет, то его надо ввести. Обычно в этом случае для каждого экземпляра сущности вводится внутренний номер, не имеющий смысла вне системы. Он называется **суррогатным ключом**.

Один и тот же набор объектов может иметь несколько ключей. Один из них является **первичным**. Это ключ, который однозначно идентифицирует отдельные экземпляры сущности. Первичный ключ должен включать в свой состав минимальное количество атрибутов. Ключ, состоящий их нескольких атрибутов, называется составным.

3. Назначение сущностям описательных атрибутов

Спецификация атрибутов заканчивается указанием для каждого атрибута множества значений, которые он может принимать. Если это множество бесконечное, то оно задается указанием типа значений (числовой, символьный и пр.) и диапазона значений для чисел и количества символов для алфавитно-цифровых значений.

4. Спецификация связей

Выявляются зависимости между двумя и более сущностями. Определяется какие из них необходимые, какие избыточные. Каждый тип связи именуется.
Рассмотрим понятие ***класс принадлежности сущности***.
Если каждый экземпляр сущности А связан с экземпляром сущности В, то класс принадлежности сущности А является обязательным. Это отмечается на ER-диаграмме черным кружком, помещённым в прямоугольник, смежный с прямоугольником сущности А.
Если не каждый экземпляр сущности А связан с экземпляром сущности В, то класс принадлежности сущности является необязательным. Это отмечается на ER-диаграмме черным кружком, помещённым на линии связи возле прямоугольника сущности А.

Правила преобразования ER-диаграмм в реляционные таблицы

Концептуальные модели позволяют более точно представить предметную область, чем реляционные и другие более ранние модели. Но в настоящее время существует немного СУБД, поддерживающих эти модели. На практике наиболее распространены системы, реализующие реляционную модель. Поэтому необходим метод перевода концептуальной модели в реляционную. Такой метод основывается на формировании набора предварительных таблиц из ER-диаграмм.
Для каждой сущности создается таблица. Причем каждому атрибуту сущности соответствует столбец таблицы.
Правила генерации таблиц из ER-диаграмм опираются на два основных фактора – тип связи и класс принадлежности сущности. Изложим их.
Правило 1: Если связь типа 1:1 и класс принадлежности обеих сущностей является обязательным, то необходима только одна таблица. Первичным ключом этой таблицы может быть первичный ключ любой из двух сущностей.
Правило 2: Если связь типа 1:1 и класс принадлежности одной сущности является обязательным, а другой - необязательным, то необходимо построить таблицу для каждой сущности. Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Первичный ключ сущности, для которой класс принадлежности является необязательным, добавляется как атрибут в таблицу сущности с обязательным классом принадлежности.

Правило 3: Если связь типа 1:1 и класс принадлежности обеих сущностей необязательный, то необходимо построить три таблицы - по одной для каждой сущности и одну для связи. Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Таблица для связи среди своих атрибутов должна иметь ключи обеих сущностей.
Правило 4: Если связь типа 1:M и класс принадлежности сущности на стороне M является обязательным, то необходимо построить таблицу для каждой сущности. Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Первичный ключ сущности на стороне 1 добавляется как атрибут в таблицу для сущности на стороне M.
Правило 5: Если связь типа 1:M и класс принадлежности сущности на стороне M является необязательным, то необходимо построить три таблицы - по одной для каждой сущности и одну для связи. Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Таблица для связи среди своих атрибутов должна иметь ключи обеих сущностей.
Правило 6: Если связь типа M:N, то необходимо построить три таблицы - по одной для каждой сущности и одну для связи. Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Таблица для связи среди своих атрибутов должна иметь ключи обеих сущностей.

Вопросы для самоконтроля:

1. Что называется концептуальной моделью?
2. Какие базовые понятия используются на этапе концептуального проектирования?
3. Какие задачи решаются на этапе концептуального проектирования?
4. Перечислите шаги концептуального проектирования.
5. Что называется сущностью и экземпляром сущности?
6. Что называется атрибутом сущности и экземпляром атрибута?
7. Что называется связью между сущностями?
8. Дайте определение понятию «класс принадлежности сущности».
9. На какие факторы опираются правила генерации таблиц из ER-диаграмм?
10. Опишите типовую пошаговую процедуру преобразования диаграммы «сущность - связь» в реляционную схему базы данных.

Нормализация БД

План:

1. Понятие нормализации
2. Первая нормальная форма
3. Вторая нормальная форма
4. Третья нормальная форма
5. Высшие нормальные формы

При проектировании БД могут появиться нежелательные свойства, такие как избыточность, аномалии обновления, аномалии включения, аномалии удаления и др. Для уменьшения нежелательных характеристик БД к схемам отношений применяют процедуры нормализации.
Нормализация - это разбиение таблицы на две или более, обладающие лучшими свойствами при включении, изменении и удалении данных.
Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором каждый факт появляется лишь в одном месте, т.е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных.
Нормализация – это процесс разделения информации на структурные единицы, т.е. таблицы.
Нормализация БД должна быть выполнена с учётом следующего правила: таблицы, которые содержат повторяющуюся информацию, для устранения дублирования значений должны быть разделены на отдельные таблицы, что приводит к сокращению размеров БД.
В теории реляционных баз данных вводятся понятия так называемых “нормальных форм” — требований к организации данных в таблицах. Нормальные формы нумеруются последовательно, по мере ужесточения требований. В правильно спроектированной БД таблицы находятся как минимум в третьей нормальной форме.
Так же можно сказать, что процесс нормализации представляет собой приведение таблиц к требуемому уровню нормальности: первый, второй и третий. Каждый уровень нормальности соответствует определённой нормальной форме.
Теория нормализации основана на концепции нормальных форм. Говорят, что таблица находится в данной нормальной форме, если она удовлетворяет определенному набору требований. Теоретически существует пять нормальных форм, но на практике обычно используются только первые три. Первые две нормальные формы являются промежуточными шагами для приведения базы данных к третьей нормальной форме.

Первая нормальная форма (1НФ)

Первая нормальная форма предписывает, что все данные, содержащиеся в таблице, должны быть атомарными (неделимыми). Перечень соответствующих атомарных типов данных определяется СУБД. Требование 1НФ совершенно естественное. Оно означает, что в каждом поле каждой записи должна находиться только одна величина, но не массив и не какая-либо другая структура данных.

Вторая нормальная форма (2НФ)

Говорят, что таблица находится во второй нормальной форме, если она находится в 1НФ и каждый не ключевой столбец полностью зависит от первичного ключа. Другими словами, значение каждого поля должно полностью определяться значением первичного ключа. Важно отметить, что

зависимость от первичного ключа понимается именно как зависимость от ключа целиком, а не от отдельной его составляющей (в случае составного ключа).
Чтобы перейти от первой нормальной формы ко второй, нужно выполнить следующую последовательность действий:

- Определить, на какие части можно разбить первичный ключ, так чтобы некоторые из неключевых полей зависели от одной из этих частей (эти части не обязаны состоять из одной колонки).
- Создать новую таблицу для каждой такой части ключа и группы, зависящих от нее полей и переместить их в эту таблицу. Часть бывшего первичного ключа станет при этом первичным ключом новой таблицы.
- Удалить из исходной таблицы поля, перемещенные в другие таблицы, кроме тех, которые станут внешними ключами.

Третья нормальная форма (3НФ)

Говорят, что таблица находится в 3НФ, если она соответствует 2НФ и все не ключевые столбцы взаимно независимы.
Взаимную зависимость столбцов удобно понимать следующим образом: столбцы являются взаимно зависимыми, если нельзя изменить один из них, не изменяя другой.
Чтобы перейти от второй нормальной формы к третьей, нужно выполнить следующую последовательность действий:

- Определить все поля, от которых зависят другие поля.
- Создать новую таблицу для каждого такого поля или группы полей и группы зависящих от него полей и переместить их в эту таблицу. Поле или группа полей, от которого зависят перемещенные поля, станет при этом первичным ключом новой таблицы.
- Удалить из исходной таблицы поля, перемещенные в другие таблицы, кроме тех, которые станут внешними ключами.

Высшие нормальные формы

В теории реляционных баз данных рассматриваются и формы высших порядков — нормальная форма Бойса — Кодда, 4НФ, 5НФ и даже выше. Большого практического значения эти формы не имеют, и разработчики, как правило, всегда останавливаются на 3НФ.

Вопросы для самоконтроля:

1. Назовите цели нормализации.
2. Чем опасно избыточное дублирование информации?
3. Назовите основные свойства нормальных форм.
4. Какие ограничения таблиц относят к 1НФ, 2НФ и 3НФ?
5. Приведите примеры таблиц, соответствующих и не соответствующих требованиям нормальных форм.

Лекция 4. Проектирование структур баз данных

План:

1. Классификация СУБД
2. Требования к СУБД
3. Общая характеристика и классификация CASE-средств
4. Основные характеристики и возможности СУБД Access
5. Типы данных СУБД Access
6. Создание новой базы данных

Классификация СУБД

Классифицировать СУБД можно по следующим признакам:

- по используемой модели данных (классификация МД была рассмотрена выше),
- по способу организации БД (централизованная или распределенная);
- по реализуемым режимам работы (однопользовательский, многопользовательский и т.д.);
- по способам физической организации данных.

Требования к СУБД

Выбор СУБД является одним из важных этапов при разработке приложений баз данных. Выбранный программный продукт должен удовлетворять как текущим, так и будущим потребностям предприятия, при этом следует учитывать финансовые затраты на приобретение необходимого оборудования, самой системы, разработку необходимого программного обеспечения на ее основе, а также обучение персонала. Кроме того, необходимо убедиться, что новая СУБД способна принести предприятию реальные выгоды.

Очевидно, наиболее простой подход при выборе СУБД основан на оценке того, в какой мере существующие системы удовлетворяют основным требованиям создаваемого проекта информационной системы. Более сложным и дорогостоящим вариантом является создание испытательного проекта на основе нескольких СУБД и последующий выбор наиболее подходящего. Но и в этом случае используются определенные критерии отбора.

Перечень требований к СУБД может изменяться в зависимости от поставленных целей. Тем не менее, можно выделить несколько групп критериев:

- реализуемые режимы работы с БД и максимальное число пользователей одновременно обращающихся к базе;
- модель данных (предусмотренные типы данных, средства поиска, реализация языка запросов, средства поддержания целостности базы данных);
- особенности архитектуры и функциональные возможности (масштабируемость, которая определяет, сможет ли данная СУБД соответствовать росту информационной системы, распределенность, сетевые возможности);

- контроль работы системы (возможность управления использованием памяти, возможность самоконфигурирования, самодиагностики производительности);
- особенности разработки приложений (средства проектирования, поддержка большого количества национальных языков, возможности разработки Web-приложений, поддерживаемые языки программирования);
- производительность, т.е. отношение количества запросов, обрабатываемых за некий промежуток времени, к стоимости всей системы, возможности параллельной обработки данных, возможности оптимизирования запросов);
- надежность (сохранность информации при сбоях, обеспечение защиты данных от несанкционированного доступа);
- требования к рабочей среде (минимальные требования к оборудованию, максимальный размер адресуемой памяти, операционные системы, под управлением которых способна работать СУБД);
- требуемый уровень квалификации персонала;
- смешанные критерии (качество и полнота документации, стоимость, стабильность производителя, распространенность СУБД).

Общая характеристика и классификация CASE-средств

Проектирование ИС - это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации специалистов. Однако до недавнего времени проектирование ИС выполнялось в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ИС. Применение структурной методологии проектирования при неавтоматизированной (ручной) разработке затруднено.

Это способствовало появлению программно-технологических средств, реализующих CASE-технологию (Computer Aided Software Engineering) создания и сопровождения ИС. Под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного программного обеспечения (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы.

Современные CASE-средства охватывают обширную область поддержки многочисленных технологий проектирования ИС: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл ПО.

Наиболее трудоемкими этапами разработки ИС являются этапы анализа и проектирования, в процессе которых CASE-средства обеспечивают качество принимаемых технических решений и подготовку проектной документации. При этом большую роль играют методы визуального представления информации. Это предполагает построение структурных или иных диаграмм в реальном масштабе времени, использование многообразной цветовой палитры, сквозную проверку синтаксических правил. Графические средства моделирования предметной области позволяют разработчикам в наглядном виде

изучать существующую ИС, перестраивать ее в соответствии с поставленными целями и имеющимися ограничениями.

Современный рынок программных средств насчитывает около 300 различных CASE-средств. Это как относительно дешевые системы для персональных компьютеров с ограниченными возможностями, так и дорогостоящие системы для неоднородных вычислительных платформ и операционных сред.

Обычно к CASE-средствам относят любое программное средство, автоматизирующее ту или иную совокупность процессов жизненного цикла программного обеспечения.

Все современные CASE-средства могут быть классифицированы в основном по типам и категориям.

Классификация по типам отражает функциональную ориентацию CASE-средств на те или иные процессы ЖЦ и включает следующие основные типы:

- средства анализа, предназначенные для построения и анализа моделей предметной области;
- средства анализа и проектирования, использующиеся для создания проектных спецификаций. Выходом таких средств являются спецификации компонентов и интерфейсов системы, архитектуры системы, алгоритмов и структур данных;
- средства проектирования баз данных, обеспечивающие моделирование данных и генерацию схем баз данных для наиболее распространенных СУБД;
- средства разработки приложений;
- средства реинжиниринга, обеспечивающие анализ программных кодов и схем баз данных и формирование на их основе различных моделей и проектных спецификаций;
- средства планирования и управления проектом;
- средства конфигурационного управления;
- средства тестирования;
- средства документирования.

Классификация по категориям определяет степень интегрированности по выполняемым функциям и включает в себя:

- отдельные локальные средства, решающие небольшие автономные задачи (tools),
- частично интегрированные средства, охватывающие большинство этапов жизненного цикла ИС (toolkit)
- полностью интегрированные средства, поддерживающие весь ЖЦ ИС и связанные общим репозиторием.

Помимо этого, CASE-средства можно классифицировать по следующим признакам:

- применяемым методологиям и моделям систем и БД;
- степени интегрированности с СУБД;
- доступным платформам.

Основные характеристики и возможности СУБД Access

СУБД Access (фирма Microsoft) имеет достаточно высокие скоростные характеристики и входит в состав чрезвычайно популярного в нашей стране и за рубежом пакета Microsoft Office. Набор команд и функций, предлагаемых

разработчикам программных продуктов в среде Access, по мощи и гибкости отвечает большинству современных требований к представлению и обработке данных. В Access 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 Basic for Applications (VBA), хорошо работает в сети.
СУБД Access позволяет использовать другие компоненты пакета Microsoft Office, такие как текстовый процессор Word for Windows, электронные таблицы Excel и т.д.
Приведем некоторые из средств Microsoft Access, существенно упрощающие разработку приложений.

1. **Процедуры обработки событий и модули форм и отчетов.** На встроенном языке VBA 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 VBA 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.
3. **Модель событий.** Модель событий, похожая на используемую в языке Microsoft Visual Basic, 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.
6. **Улучшенные средства отладки.** Помимо установки точек прерывания и пошагового выполнения программ на языке VBA, можно вывести на экран список всех активных процедур.
7. **Процедура обработки ошибок.** Помимо традиционных способов обработки ошибок возможно использование процедуры обработки события Error для 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 VBA можно создавать программы-надстройки, например нестандартные мастера и построители. Мастер — средство Microsoft Access, 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 Microsoft Access.
11. **Мастера Access.** Access позволяет даже мало подготовленному пользователю создать свою БД, обрабатывать данные с помощью форм, запросов и отчетов, проводить анализ таблиц БД и выполнять ряд других работ. Практически для любых работ с БД в Access имеется свой мастер, который помогает их выполнять.

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.

Типы данных базы данных Microsoft Access

Тип	Содержимое типа данных
Текстовый	Текст и числа, например, имена и адреса, номера телефонов и почтовые индексы. Текстовое поле может содержать до 255 символов
Поле Memo	Длинный текст и числа, например комментарии и пояснения. Поле Memo может содержать до 64000 символов
Числовой	Числовые данные, допускающие проведение математических расчетов, за исключением денежных. Свойство *Размер поля* (FieldSize) позволяет указывать различные типы числовых данных
Дата/время	Значения даты и времени. Пользователь имеет возможность выбора одного из многочисленных стандартных форматов или создания специального формата
Денежный	Денежные значения (не рекомендуется использовать для проведения денежных расчетов значения, принадлежащие к числовому типу данных, так как последние могут округляться при расчетах), которые всегда выводятся с указанным числом десятичных знаков после запятой
Счетчик	Автоматически вставляющиеся последовательные номера. Нумерация начинается с единицы. Поле счетчика, удобное для создания ключа, является совместимым с полем числового типа, для которого в свойстве *Размер поля* (FieldSize) указано значение *Длинное целое*
Логический	Значения *Да/Нет, Истина/Ложь, Вкл./Выкл.*
Поле объекта OLE	

Объекты, созданные в других программах, поддерживающих протокол OLE, которые связываются или внедряются в базу данных Microsoft Access через элемент управления в форме или отчете

Создание новой базы данных

Создание новой базы данных Access осуществляется в соответствии с ее структурой, полученной в результате внемашинного проектирования, заключающегося в создании информационно-логической модели предметной области. Структура реляционной базы данных определяется составом таблиц и их взаимосвязями. Создание реляционной базы данных с помощью СУБД Access на компьютере начинается с формирования структуры таблиц. При этом формируется состав полей и задается их описание.
После формирования структуры таблиц создается схема данных, в которой устанавливаются связи между таблицами. Access запоминает и использует эти связи при заполнении таблиц и обработке данных. Завершается создание базы данных процедурой заполнения таблиц конкретной информацией.
После запуска MS Access одновременно с окном базы данных открывается первое диалоговое окно, позволяющее начать создание БД или открыть уже существующую. На закладках (кнопках) окна базы данных представлены основные типы ее объектов: *Таблицы, Запросы, Формы, Отчеты, Макросы, Модули.*

Вопросы для самоконтроля:

1. Приведите классификацию СУБД по различным признакам.
2. Какие требования предъявляются к СУБД?
3. Дайте определение CASE-средствам и CASE-технологии.
4. Назовите признаки классификации CASE-средств.
5. Дайте краткую характеристику СУБД Access.
6. Какие типы данных поддерживаются СУБД Access?

Организация интерфейса с пользователем

План:

1. Основные требования к разработке пользовательского интерфейса
2. Основы создания формы
3. Элементы управления

Основные требования к разработке пользовательского интерфейса

Создание пользовательского приложения требует разработки так называемого дружественного интерфейса пользователя, т.е. организации диалога между пользователем и компьютером (клиентом и сервером).

Основным способом организации диалога является разработка диалоговых форм, которые по назначению можно подразделить на следующие группы:

- для ввода данных в таблицы;
- для ввода условий обработки информации в запросы;
- для автоматизации работы с объектами базы данных.

Формы для ввода данных в таблицы предназначаются для такой организации процедур внесения информации, которые могли бы свести к минимуму возможность ошибок оператора. Кроме того, такие формы могут служить для проведения анализа имеющихся в таблицах данных.

Формы для ввода условий обработки информации в запросы имеют назначение, аналогичное формам для ввода данных в таблицы.

Формы для автоматизации работы с объектами базы данных имеют различное назначение, например это формы-заставки, формы-меню, кнопочные формы и др.

Все эти формы и представляют собой интерфейс пользователя.

Разработка форм может производиться различными средствами визуального проектирования, например:

- с помощью языков программирования (C++, Delphi, VBA);
- с помощью специальных компонентов СУБД (конструкторов форм Microsoft Access, Oracle и др.).

Однако какими бы средствами не разрабатывались формы интерфейса пользователя, необходимо учитывать следующие советы и рекомендации:

- прежде чем приступать к проектированию форм, необходимо продумать «сценарий» пользовательского интерфейса, т.е. определить последовательность появления форм на экране компьютера пользователя в соответствии с выполняемыми задачами. Фактически разработчик форм должен научиться создавать сценарии аналогично сценаристу художественных фильмов;

- каждая форма должна иметь название, которое однозначно определяет ее назначение;
- форма должна иметь привлекательный внешний вид, но при этом не должна содержать информации, не относящейся к конкретной задаче;
- формы для ввода данных в таблицы или параметров в запросах должны обеспечивать: минимизацию возможных ошибок при вводе данных пользователем за счет согласования терминов и сокращений, ввода данных из списков и создания сообщений о допущенной ошибке;
- оптимальные способы перемещения курсора (табуляцией, стрелками, указателем мыши); получение пояснительных сообщений или инструкций при вводе данных в поля таблиц или запросов; автоматическое закрытие формы и переход к следующей форме.

Основы создания формы

Однотабличная форма может быть создана пользователем в режиме **Конструктора** форм или с помощью **Мастера**. В первом случае создание начинается с пустой формы и конструирование полностью возлагается

на пользователя. Для создания однотабличной формы целесообразно использовать **Мастер форм** или команды **Автоформа**.

Чтобы начать создание формы, надо в окне базы данных выбрать закладку **Формы** и нажать кнопку **Создать**. Открывающееся диалоговое окно **Новая форма** представляет возможность выбрать один из режимов создания формы:

- Конструктор;
- Мастер форм;
- Автоформа: в столбец;
- Автоформа: ленточная;
- Автоформа: табличная;
- Диаграмма;
- Сводная таблица

Формы, которые удовлетворяют любому, даже самому требовательному вкусу, можно создать с помощью Конструктора.

Эффективным способом работы является быстрое создание форм с помощью Мастера форм и дальнейшее их совершенствование с помощью Конструктора.

Мастер форм может создавать форму для одной таблицы и для нескольких взаимосвязанных таблиц.

При выборе только одной таблицы могут быть созданы формы:

- В один столбец;
- Ленточная;
- Табличная.

Форма **В один столбец** выводит в виде колонок для просмотра данные только одной записи, поля которой расположены в нужном порядке.

Ленточная форма выводит одну и более записей в зависимости от того, сколько можно уместить их на экране.

Табличная форма выводит данные обычным табличным способом, но в отличие от таблиц может выбирать поля для вывода.

Мастер форм позволяет пользователю определить, какие поля таблицы включаются в форму, и выбрать стиль ее оформления. Выбор таблицы для создания формы может быть произведен как в окне **Новая форма**, так и в первом диалоговом окне мастера **Создание форм**.

Команды **Автоформа: в столбец**, **Автоформа: ленточная** и **Автоформа: табличная** создают для заданной таблицы формы, которые отличаются от форм, создаваемых мастером, тем, что включают все поля таблицы и не предоставляют возможности выбора стиля оформления. Эти команды, не вступая в диалог с пользователем и не отображая формы в режиме конструктора, выводят ее на экран в режиме формы, то есть заполненную значениями из таблицы.

Заметим, что таблица, для которой строится форма, выбирается в окне **Новая форма**. Форма, созданная мастером, также как и форма, созданная любой командой **Автоформа**, может быть отредактирована в соответствии с требованиями пользователя. Редактирование выполняется в режиме **Конструктора** форм.

Последние опции — **Сводная таблица** и **Диаграмма** — позволяют создавать достаточно специализированные по своим задачам формы и активно используют OLE-технологии.

Элементы управления

Все сведения в форме или отчете содержатся в элементах управления. Элементы управления — это объекты формы или отчета, которые служат для вывода данных на экран, выполнения макрокоманд или оформления формы или отчета.

Например, поле можно использовать для вывода данных на экран в форме или отчете, кнопку — для открытия другой формы или отчета, а линию или прямоугольник — для разделения и группировки элементов управления с тем, чтобы они лучше воспринимались пользователем.

В Microsoft Access существуют следующие типы элементов управления, которые содержатся на панели элементов в режиме конструктора формы или режиме конструктора запроса:

- надпись;
- поле;
- группа;
- выключатель;
- переключатель;
- флажок;
- поле со списком;
- список;
- кнопка;
- рисунок;
- свободная рамка объекта;
- присоединенная рамка объекта;
- разрыв страницы;
- набор вкладок;
- подчиненная форма/отчет;
- линия;
- прямоугольник;
- и дополнительные элементы ActiveX.

Элементы управления могут быть

- связанными;
- свободными;
- вычисляемыми.

Связанный элемент управления присоединен к полю базовой таблицы или запроса. Такие элементы управления используются для отображения, ввода или обновления значений из полей базы данных.

Для **вычисляемого элемента** управления в качестве источника данных используется выражение. В выражении могут быть использованы данные из поля базовой таблицы или запроса для формы или отчета, а также данные другого элемента управления формы или отчета.

Для **свободного элемента** управления источника данных не существует. Свободные элементы управления используются для вывода на экран данных, линий, прямоугольников и рисунков.

Вопросы для самоконтроля:

1. Какой режим представления данных обеспечивает максимальную гибкость для просмотра и ввода данных?
2. Какие действия можно выполнять, работая с формой?
3. Перечислите основные типы форм.
4. Перечислите способы создания форм.

Лекция 5. Организация запросов на выборку данных при помощи языка SQL

План:

1. Появление языка SQL.
2. Типы команд SQL
3. Преимущества языка SQL

Рост количества данных, необходимость их хранения и обработки привели к тому, что возникла потребность в создании *стандартного языка баз данных*, который мог бы функционировать в многочисленных компьютерных системах различных видов. Действительно, с его помощью пользователи могут манипулировать данными независимо от того, работают ли они на персональном компьютере, сетевой рабочей станции или универсальной ЭВМ. Одним из языков, появившихся в результате разработки реляционной модели данных, является язык SQL (Structured Query Language), который в настоящее время получил очень широкое распространение и фактически превратился в *стандартный язык реляционных баз данных*. Стандарт на язык SQL был выпущен Американским национальным институтом стандартов (ANSI) в 1986 г., а в 1987 г. Международная организация стандартов (ISO) приняла его в качестве международного.

Типы команд SQL

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволила создать компактный язык с небольшим набором предложений. Язык SQL может использоваться как для выполнения *запросов* к данным, так и для построения прикладных программ. Основные *категории команд* языка SQL предназначены для выполнения различных функций, включая построение объектов *базы данных* и манипулирование ими, начальную загрузку данных в *таблицы*, обновление и

удаление существующей информации, выполнение *запросов* к *базе данных*, управление доступом к ней и ее общее администрирование.

Основные *категории команд* языка SQL:

- DDL – язык определения данных;
- DML – язык манипулирования данными;
- DQL – язык *запросов*;
- *DCL* – язык управления данными;
- команды администрирования данных;
- команды управления транзакциями

Определение структур базы данных (DDL)

Язык определения данных (Data Definition Language, DDL) позволяет создавать и изменять структуру объектов *базы данных*, например, создавать и удалять *таблицы*.

Основными командами языка DDL являются следующие: **CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.**

Манипулирование данными (DML)

Язык манипулирования данными (*Data Manipulation Language*, DML) используется для манипулирования информацией внутри объектов *реляционной базы данных* посредством трех основных команд: **INSERT,UPDATE, DELETE.**

Выборка данных (DQL)

Язык *запросов* DQL наиболее известен пользователям *реляционной базы данных*, несмотря на то, что он включает всего одну команду **SELECT.** Эта команда вместе со своими многочисленными опциями и предложениями используется для формирования *запросов* к *реляционной базе данных*.

Язык управления данными (DCL - Data Control Language)

Команды управления данными позволяют управлять доступом к информации, находящейся внутри *базы данных*. Как правило, они используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями. Команды управления данными следующие: **GRANT, REVOKE.**

Команды администрирования данных

С помощью команд администрирования данных пользователь осуществляет контроль над выполняемыми действиями и анализирует операции *базы данных*; они также могут оказаться полезными при анализе производительности системы. Не следует путать администрирование данных с администрированием *базы данных*, которое представляет собой общее управление *базой данных* и подразумевает использование команд всех уровней.

Команды управления транзакциями

Существуют следующие команды, позволяющие управлять транзакциями *базы данных*: **COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.**

Преимущества языка SQL

Язык SQL является основой многих *СУБД*, т.к. отвечает за физическое структурирование и запись данных на диск, а также за чтение данных с диска, позволяет принимать SQL- *запросы* от других компонентов *СУБД* и

пользовательских приложений. Таким образом, SQL – мощный инструмент, который обеспечивает пользователям, программам и вычислительным системам доступ к информации, содержащейся в *реляционных базах данных*.

Основные достоинства языка SQL заключаются в следующем:

- стандартность;
- независимость от конкретных *СУБД*;
- возможность переноса с одной вычислительной системы на другую;
- реляционная основа языка;
- возможность создания *интерактивных запросов*;
- возможность программного доступа к БД;
- обеспечение различного представления данных;
- возможность динамического изменения и расширения структуры БД;
- поддержка архитектуры *клиент-сервер*.

Любой язык работы с *базами данных* должен предоставлять пользователю следующие возможности:

- создавать *базы данных* и *таблицы* с полным описанием их структуры;
- выполнять основные операции манипулирования данными, в частности, вставку, модификацию и удаление данных из *таблиц*;
- выполнять простые и сложные *запросы*, осуществляющие преобразование данных.

Кроме того, язык работы с *базами данных* должен решать все указанные выше задачи при минимальных усилиях со стороны пользователя, а структура и синтаксис его команд – достаточно просты и доступны для изучения. И наконец, он должен быть универсальным, т.е. отвечать некоторому признанному *стандарту*, что позволит использовать один и тот же синтаксис и структуру команд при переходе от одной *СУБД* к другой. Язык SQL удовлетворяет практически всем этим требованиям.

Вопросы для самоконтроля:

1. Дайте определение понятию SQL.
2. Перечислите категории команд языка SQL.
3. В чем состоит основное достоинство SQL?

Синтаксис операторов, типы данных

План:

1. Синтаксис SQL-операторов
2. Типы данных SQL

Синтаксис SQL-операторов

Оператор SQL состоит из *зарезервированных слов*, а также из слов, определяемых пользователем. *Зарезервированные слова* являются постоянной частью языка SQL и имеют фиксированное значение. Их следует записывать в

точности так, как это установлено, нельзя разбивать на части для переноса с одной *строки* на другую. Слова, определяемые пользователем, задаются им самим (в соответствии с синтаксическими правилами) и представляют собой идентификаторы или имена различных объектов базы данных. Слова в операторе размещаются также в соответствии с установленными синтаксическими правилами.

Идентификаторы языка SQL предназначены для обозначения объектов в базе данных и являются именами таблиц, представлений, столбцов и других объектов базы данных. Символы, которые могут использоваться в создаваемых пользователем идентификаторах языка SQL, должны быть определены как набор символов. Стандарт SQL задает набор символов, который используется по умолчанию, – он включает строчные и прописные буквы латинского алфавита (**A-Z, a-z**), цифры (**0-9**) и символ подчеркивания (_). На формат идентификатора накладываются следующие ограничения:

- *идентификатор* может иметь длину до 128 символов;
- *идентификатор* должен начинаться с буквы;
- *идентификатор* не может содержать пробелы.

Большинство компонентов языка не чувствительны к регистру. Поскольку у языка SQL свободный формат, отдельные SQL-операторы и их последовательности будут иметь более читаемый вид при использовании отступов и выравнивания.

Язык, в терминах которого дается описание языка SQL, называется ***метаязыком***. Синтаксические определения обычно задают с помощью специальной металингвистической символики, называемой ***Бэкуса-Науэра формулами*** (БНФ). Прописные буквы используются для записи зарезервированных слов и должны указываться в операторах точно так, как это будет показано. Строчные буквы употребляются для записи слов, определяемых пользователем. Применяемые в нотации *БНФ* символы и их обозначения показаны в таблице:

Символ	Обозначение
Равно по определению	::=
Необходимость выбора одного из нескольких приведенных значений	|
Описанная с помощью *метаязыка* структура	<…>
Обязательный выбор некоторой конструкции из списка	{…}
Необязательный выбор некоторой конструкции из списка	[…]
Необязательная возможность повторения конструкции от нуля до нескольких раз	[,…n]

Типы данных SQL

1. Символьные типы данных - содержат буквы, цифры и специальные символы.

- **CHAR** или **CHAR(n)** -символьные строки фиксированной длины. Длина строки определяется параметром **n**. **CHAR** без параметра соответствует **CHAR(1)**. Для хранения таких данных всегда отводится **n** байт вне зависимости от реальной длины строки.
- **VARCHAR(n)** - символьная строка переменной длины. Для хранения данных этого типа отводится число байт, соответствующее реальной длине строки.

2. Целые типы данных - поддерживают только целые числа (дробные части и десятичные точки не допускаются). Над этими типами разрешается выполнять арифметические операции и применять к ним агрегирующие функции (определение максимального, минимального, среднего и суммарного значения столбца реляционной таблицы).

- **INTEGER** или **INT**- целое, для хранения которого отводится, как правило, 4 байта. *(Замечание: число байт, отводимое для хранения того или иного числового типа данных зависит от используемой СУБД и аппаратной платформы, здесь приводятся наиболее "типичные" значения).* Интервал значений от - 2147483647 до + 2147483648
- **SMALLINT** - короткое целое (2 байта), интервал значений от - 32767 до +32768

3. Вещественные типы данных - описывают числа с дробной частью.

- **FLOAT** и **SMALLFLOAT** - числа с плавающей точкой (для хранения отводится обычно 8 и 4 байта соответственно).
- **DECIMAL(p)** - тип данных аналогичный **FLOAT** с числом значащих цифр **p**.
- **DECIMAL(p,n)** - аналогично предыдущему, **p** - общее количество десятичных цифр, **n** - количество цифр после десятичной запятой.

4. Денежные типы данных - описывают, естественно, денежные величины. Если ваша система такого типа данных не поддерживает, то используйте DECIMAL(p,n).

- **MONEY** - все аналогично типу **DECIMAL(p,n)**. Вводится только потому, что некоторые СУБД предусматривают для него специальные методы форматирования.

5. Дата и время - используются для хранения даты, времени и их комбинаций. Большинство СУБД умеет определять интервал между двумя датами, а также уменьшать или увеличивать дату на определенное количество времени.

- **DATE** - тип данных для хранения даты.
- **TIME** - тип данных для хранения времени.
- **INTERVAL** - тип данных для хранения временного интервала.
- **DATETIME** - тип данных для хранения моментов времени (год + месяц + день + часы + минуты + секунды + доли секунд).

6. Двоичные типы данных - позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.). Определения этих типов наиболее сильно различаются от системы к системе, часто используются ключевые слова:

- **BINARY**
- **BYTE**
- **BLOB**

7. Последовательные типы данных - используются для представления возрастающих числовых последовательностей.

- **SERIAL** - тип данных на основе **INTEGER**, позволяющий сформировать уникальное значение (например, для первичного ключа). При добавлении записи СУБД автоматически присваивает полю данного типа значение, получаемое из возрастающей последовательности целых чисел.
  В заключение следует сказать, что для всех типов данных имеется общее значение **NULL** - "не определено". Это значение имеет каждый элемент столбца до тех пор, пока в него не будут введены данные. При создании таблицы можно явно указать СУБД могут ли элементы того или иного столбца иметь значения **NULL** (это не допустимо, например, для столбца, являющего первичным ключом).

Вопросы для самоконтроля:

1. Из каких слов состоит оператор SQL?
2. Какие ограничения накладываются на формат идентификатора?
3. Дайте определение понятию «метаязык».
4. Какие символы применяются в нотации БНФ? Что они обозначают?
5. Какие типы данных поддерживаются в SQL?

Создание, модификация и удаление таблиц

План:

1. Создание таблицы
2. Модификация таблиц
3. Удаление таблиц

Создание таблицы

```
CREATE TABLE <имя_таблицы>
(<имя_столбца> <тип_столбца>
[NOT NULL]
[UNIQUE | PRIMARY KEY]
[REFERENCES <имя_ таблицы> (<имя_столбца>)]
, ...)
```

Пользователь обязан указать имя таблицы и список столбцов. Для каждого столбца обязательно указываются его имя и тип, а также опционально могут быть указаны параметры:

- **NOT NULL** - в этом случае элементы столбца всегда должны иметь определенное значение (не NULL)
- один из взаимоисключающих параметров **UNIQUE** - значение каждого элемента столбца должно быть уникальным или **PRIMARY KEY** - столбец является первичным ключом.
- **REFERNCES <имя_мастер_таблицы> [<имя_столбца>]** - эта конструкция определяет, что данный столбец является внешним ключом и указывает на ключ какой мастер_таблицы он ссылается.

Контроль за выполнением указанных условий осуществляет СУБД

Модификация таблиц

Как бы тщательно не планировалась структура таблицы, иногда возникает необходимость внести в нее некоторые изменения. Предположим, что в уже сформированную таблицу необходимо добавить столбец. Эту операцию можно выполнять различными путями. Например, можно удалить таблицу со старой структурой и создать вместо нее новую таблицу с нужной структурой. Недостатком этого метода является то, что необходимо будет куда-то скопировать имеющиеся в таблице данные и переписать их в новую таблицу после ее создания.

Специальная команда ALTER TABLE предназначена для модификации структуры таблицы. С ее помощью можно изменять свойства существующих столбцов, удалять или добавлять в таблицу столбцы, а также управлять ограничением целостности, как на уровне столбца, так и на уровне таблицы, т.е. выполнять следующие функции:

-
  - добавить в таблицу определение нового столбца;
  - удалить столбец из таблицы;
  - изменить значение по умолчанию для какого-либо столбца;
  - добавить или удалить первичный ключ таблицы;
  - добавить или удалить внешний ключ таблицы;
  - добавить или удалить условие уникальности;
  - добавить или удалить условие на значение.

Команда ALTER TABLE берет на себя все действия по копированию данных во временную таблицу, удалению старой таблицы и созданию вместо нее новой таблицы с нужной структурой и последующим переписыванием в нее данных. Назначение многих параметров и ключевых слов команды ALTER TABLE аналогично назначению соответствующих параметров и ключевых слов команды CREATE TABLE.

Рассмотрим основные режимы использования команды ALTER TABLE:

- добавление столбца;
- удаление столбца;
- модификация столбца;

1. **Добавление столбца**

```
ALTER TABLE <имя_таблицы> ADD
(<имя_столбца> <тип_столбца>
[NOT NULL]
[UNIQUE | PRIMARY KEY]
[REFERENCES <имя_мастер_таблицы> (<имя_столбца>)]
,...)
```

2. **Модификация столбца**

```
ALTER TABLE <имя_таблицы>
ALTER COLUMN(<имя_столбца> <тип_столбца>
[NOT NULL]
   [UNIQUE | PRIMARY KEY]
   [REFERENCES <имя_мастер_таблицы> (<имя_столбца>)]
   ,...)
```

Изменение столбца невозможно, если:

- столбец участвует в ограничениях PRIMARY KEY или FOREIGN KEY;
- на столбец наложены ограничения целостности, например UNIQUE (исключение – столбцы, имеющие тип данных переменной длины;
- со столбцом связано значение по умолчанию.

Определяя для столбца новый тип данных, следует помнить о том, что старый тип данных должен конвертироваться в новый.

3. **Удаление столбца**

```
ALTER TABLE <имя_таблицы> DROP
(<имя_столбца>
,...)
```

Нельзя удалять столбцы с ограничением целостности CHECK, FOREIGN KEY, UNIQUE или PRIMARY KEY, а также столбцы, для которых определены значения по умолчанию.

Удаление таблиц

```
DROP TABLE <имя_таблицы>
```

Невозможно удалить таблицу, если на нее ссылается другая таблица.

Вопросы для самоконтроля:

1. Приведите общий синтаксис SQL-оператора для создания таблицы.
2. Приведите общий синтаксис SQL-оператора для добавления столбца в таблицу.

3. Приведите общий синтаксис SQL-оператора для модификации столбца.
4. В каких случаях модификация столбца невозможна?
5. Приведите общий синтаксис SQL-оператора для удаления столбца.
6. В каких случаях удаление столбца невозможно?

Операторы манипулирования данными

План:

1. Добавление новой записи в таблицу
2. Модификация записей
3. Удаление записей

К этой группе относятся операторы добавления, изменения и удаления записей.

1. **Добавление новой записи в таблицу**

```
INSERT INTO <имя_таблицы>
[<имя_столбца>,<имя_столбца>,..)]
VALUES (<значение>,<значение>,..);
```

Список столбцов в данной команде не является обязательным параметром. В этом случае должны быть указаны значения для всех полей таблицы в том порядке, как эти столбцы были перечислены в команде `CREATE TABLE`.

2. **Модификация записей**

```
UPDATE <имя_таблицы> SET <имя_столбца>=<значение>,...
WHERE <условие>]
```

Если задано ключевое слово `WHERE` и условие, то команда `UPDATE` применяется только к тем записям, для которых оно выполняется. Если условие не задано, `UPDATE` применяется ко всем записям.

В качестве условия используются логические выражения над константами и полями. В условиях допускаются:

- операции сравнения: `>` , `<` , `>=` , `<=` , `=` , `<>` , `!=` . В SQL эти операции могут применяться не только к числовым значениям, но и к строкам (`"<"` означает раньше, а `">"` позже в алфавитном порядке) и датам (`"<"` раньше и `">"` позже в хронологическом порядке).
- операции проверки поля на значение `NULL`: `IS NULL, IS NOT NULL`
- операции проверки на вхождение в диапазон: `BETWEEN` и `NOT BETWEEN`.
- операции проверки на вхождение в список: `IN` и `NOT IN`
- операции проверки на вхождение подстроки: `LIKE` и `NOT LIKE`

- отдельные операции соединяются связями `AND, OR, NOT` и группируются с помощью скобок.

3. **Удаление записей**

```
DELETE FROM <имя_таблицы>
[ WHERE <условие> ]
```

Удаляются все записи, удовлетворяющие указанному условию. Если ключевое слово `WHERE` и условие отсутствуют, из таблицы удаляются все записи.

Вопросы для самоконтроля:

1. Приведите общий синтаксис SQL-оператора для добавления записи в таблицу.
2. Почему список столбцов в данной команде не является обязательным параметром?
3. Приведите общий синтаксис SQL-оператора для модификации записи.
4. Какие операции допустимы в логических выражениях условия?
5. Приведите общий синтаксис SQL-оператора для удаления записи.

Организация запросов на выборку данных при помощи языка SQL

План:

1. Синтаксис оператора SELECT
2. Выборка из нескольких таблиц

Синтаксис оператора SELECT

Для извлечения записей из таблиц в SQL определен оператор **SELECT**. С помощью этой команды осуществляется не только операция реляционной алгебры "выборка" (горизонтальное подмножество), но и предварительное соединение двух и более таблиц. Это наиболее сложное и мощное средство SQL, полный синтаксис оператора **SELECT** имеет вид:

```
  SELECT [ALL | DISTINCT] <список_выбора>
    FROM <имя_таблицы>
    [ WHERE <условие> ]
    [ GROUP BY <имя_столбца>,... ]
    [ HAVING <условие> ]
    [ORDER BY <имя_столбца> [ASC | DESC],... ]
```

Порядок предложений в операторе **SELECT** должен строго соблюдаться (например, GROUP BY должно всегда предшествовать ORDER BY), иначе это приведет к появлению ошибок.
Этот оператор всегда начинается с ключевого слова **SELECT**.
В конструкции <список_выбора> определяется столбец или столбцы, включаемые в результат. Он может состоять из имен одного или нескольких столбцов, или из одного символа * (звездочка), определяющего все столбцы. Элементы списка разделяются запятыми.
В том случае, когда нас интересуют не все записи, а только те, которые удовлетворяют некому условию, это условие можно указать после ключевого слова WHERE.
В заключение заметим, что при выполнении оператора SELECT результирующее отношение может иметь несколько записей с одинаковыми значениями всех полей. Чтобы исключить повторяющиеся записи из выборки используется ключевое слово DISTINCT. Ключевое слово ALL указывает, что в результат необходимо включать все строки.

Выборка из нескольких таблиц

Очень часто возникает ситуация, когда выборку данных надо производить из отношения, которое является результатом слияния двух других отношений. Для выполнения операции такого рода в операторе SELECT после ключевого слова FROM указывается список таблиц, по которым производится поиск данных. После ключевого слова WHERE указывается условие, по которому производится слияние.
Следует обратить внимание на то, что когда в разных таблицах присутствуют одноименные поля, то для устранения неоднозначности перед именем поля указывается имя таблицы и знак "." (точка). **(Хорошее правило: имя таблицы указывать всегда!)**

Замечание:
Естественно, имеется возможность производить слияние и более чем двух таблиц.

Вопросы для самоконтроля:

1. Приведите общий синтаксис SQL-оператора SELECT.
2. Для чего используются ключевые слова ALL и DISTINCT?
3. Для чего используется ключевое слово FROM?
4. Для чего используются ключевое слово WHERE?
5. Как произвести выборку данных из нескольких связанных таблиц?

Сортировка и группировка данных при помощи языка SQL

План:

1. Группировка данных
2. Сортировка данных

Группировка данных

Группировка данных в операторе SELECT осуществляется с помощью ключевого слова GROUP BY и ключевого слова HAVING, с помощью которого задаются условия разбиения записей на группы.
GROUP BY неразрывно связано с агрегирующими функциями, без них оно практически не используется. GROUP BY разделяет таблицу на группы, а агрегирующая функция вычисляет для каждой из них итоговое значение.
Ключевое слово HAVING работает следующим образом: сначала GROUP BY разбивает строки на группы, затем на полученные наборы накладываются условия HAVING.

Сортировка данных

Для сортировки данных, получаемых при помощи оператора SELECT служит ключевое слово ORDER BY. С его помощью можно сортировать результаты по любому столбцу или выражению, указанному в <списке_выбора>. Данные могут быть упорядочены как по возрастанию, так и по убыванию.
Ключевое слово DESC задает обратный порядок сортировки, ключевое слов ASC (его можно опускать) - прямой порядок сортировки.

Вопросы для самоконтроля:

1. С помощью какого ключевого слова осуществляется группировка данных в операторе SELECT?
2. Для чего используется ключевое слово HAVING?
3. С помощью какого ключевого слова осуществляется сортировка данных в операторе SELECT?
4. Как отсортировать данные по возрастанию (убыванию)?

Функции в запросах SQL

План:

1. Агрегатные функции
2. Преобразование текста
3. Работа со строками

SQL позволяет выполнять различные арифметические операции над столбцами результирующего отношения. В конструкции <список_выбора> можно использовать константы, функции и их комбинации с арифметическими операциями и скобками.
В арифметических выражениях допускаются операции сложения (+), вычитания (-), деления (/), умножения (*), а также различные функции (COS, SIN, ABS - абсолютное значение и т.д.).

Агрегатные функции

В SQL также определены так называемые агрегатные функции, которые совершают действия над совокупностью одинаковых полей в группе записей. Среди них:

- **AVG(<имя поля>)** - среднее по всем значениям данного поля
- **COUNT(<имя поля>)** или **COUNT (*)** - число записей
- **MAX(<имя поля>)** - максимальное из всех значений данного поля
- **MIN(<имя поля>)** - минимальное из всех значений данного поля
- **SUM(<имя поля>)** - сумма всех значений данного поля

Следует учитывать, что каждая агрегирующая функция возвращает единственное значение.
Область действия данных функции можно ограничить с помощью логического условия.

Преобразование текста

Часто, текстовые значения заполняются пользователями программного обеспечения по-разному: кто пишет Ф.И.О. с заглавной буквы, кто нет; кто-то пишет все заглавными буквами. Многие отчетные формы требуют унифицированного подхода, да и не только отчетные формы. Для решения этой задачи в SQL есть две функции UCASE - преобразует символы строки в верхний регистр и LCASE - преобразует символы строки в нижний регистр.

Работа со строками

`MID(<text>, <start_num>, <num_chars>)` - возвращает строку символов из середины текстовой строки с учетом начальной позиции и длины, где text - текстовая строка, из которой нужно извлечь символы, или столбец, содержащий текст; start_num - положение первого символа, который необходимо извлечь (начинаются с 1); num_chars - число возвращаемых символов.
Иногда приходится в качестве аргументов функции MID использовать выражения с функцией LEN:

`LEN(column_name)` - возвращает длину значения в поле записи.
Функция LEN( ) исключает из подсчета конечные пробелы.

Вопросы для самоконтроля:

1. Какие операции и функции можно выполнять над данными в SQL?
2. Что такое агрегатные функции? Какие функции входят в эту группу?
3. Какие функции для работы со строками в SQL вам известны?