Некоммерческое акционерное общество

АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ

Кафедра информационных систем

БАЗЫ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

Конспект лекции
для студентов специальности 5В070300 – Информационные системы

Алматы 2017

СОСТАВИТЕЛЬ: А.Т. Купарова. Базы данных в информационных системах. Конспект лекций для студентов специальности 5В070300 – Информационные системы. – Алматы: АУЭС, 2017. – 74 с.

Конспект лекции по дисциплине «Базы данных в информационных системах» состоят из 15 лекций. Целью курса «Базы данных в информационных системах» является изучение основ методологии проектирования баз данных в информационных системах: концептуальному, логическому и физическому проектированию на примере иерархических, сетевых и реляционных баз данных, а также языков создания; формирование представлений об архитектуре, основных подходах к проектированию и областях применения систем баз данных.

В материалах рассматриваются основные понятия из области баз данных, описываются функции банка данных и СУБД, свойства данных, поддерживаемые в базе, используемые языки, уровни представления данных.

Конспект лекций предназначен для студентов специальности 5В070300 - Информационные системы.

Ил.8, табл.11, библиогр. – 10 назв.

Рецензент: к.т.н., доцент Матаев У.М.

Печатается по издания некоммерческого акционерного общества «Алматинский университет энергетики и связи» на 2017 г.

Дополнительный план 2017 г., поз.3

Айжан Токжумаевна Купарова

БАЗЫ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

Конспект лекций
для студентов специальности 5В070300 –Информационные системы

Редактор Л.Т. Сластихина
Специалист по стандартизации Н.К. Молдабекова

Подписано в печать . Формат 60х84 1/16
Тираж 20 экз. Бумага типографская №1
Объем 4,5 уч.-изд. л. Заказ . Цена 2300 тенге.

Копировально-множительное бюро
некоммерческого акционерного общества
«Алматинский университет энергетики и связи»
050013 Алматы, Байтурсынова, 126

Некоммерческое акционерное общество
АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
Кафедра «Информационные системы»

УТВЕРЖДАЮ

Проректор по учебно-методической работе
_____С. В. Коньшин
«____»_____2017 г.

Базы данных в информационных системах
для специальности 5В070300 – Информационные системы

Конспект лекций

СОГЛАСОВАНО
Директор УМД
_____Р.Р. Мухамеджанова
«____» _____ 2017 г.

Председатель УМК
_____Б.К. Курпенов
_____2017 г.

Редактор
_____ _____

«___» _____2017 г.

Специалист по стандартизации
_____ _____
«___» _____ 2017 г.

Рассмотрено и одобрено на
заседаний кафедры ИС
Протокол № __от «__» _____2017 г.
Зав. кафедрой ИС
_____ Т.С. Картбаев

Составитель
_____ А.Т. Купарова

Алматы 2017

Содержание

Введение

Исторически сложившееся развитие вычислительных систем обусловило необходимость хранения в электронном (машиночитаемом) виде все большего количества информации. Во всех областях человеческой деятельности используется такое понятие, как информация, так как любая взаимосвязь и координация работ возможны только благодаря информации. Проблеме создания различных средств и методов обработки информации всегда уделялось большое внимание. Человек создал естественные информационные системы, поскольку существовала насущная потребность снабжать производство информацией, необходимой при контроле и принятии решений, научился собирать эту информацию, обрабатывать и передавать её по назначению. Тем более, что в условиях рыночной экономики информация выступает как один из важнейших товаров.

Сложности, возникшие при решении на практике задач структурированного хранения и эффективной обработки возрастающих объемов информации, стимулировали исследования в соответствующих областях. Задачи хранения и обработки данных были формализованы. Была создана теоретическая база для решения задач такого класса, результатом реализации на практике, которой стали системы, предназначенные для организации обработки, хранения и предоставления доступа к информации. Позже такие системы стали называть системами баз данных.

Основные идеи современных технологий базируются на концепции баз данных. Основой любой информационной технологии и информационной системы являются данные, которые должны быть организованы в базы данных с целью адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей. Расширение объема и структурной сложности хранимых данных, увеличение круга пользователей информационных систем привело к созданию систем управления базами данных (СУБД), предназначенных для организации и ведения баз данных.

Целью курса является изучение теоретических основ построения баз данных (БД) в информационных системах (ИС), методов организации поиска и обработки данных в ИС, языковых средствах описания и манипулирования данными, принципов построения основных моделей данных и их использование в современных системах управления базами данными СУБД.

Обсуждается вопросы защиты и безопасности информации в базах данных и информационных системах, стандартного языка управления данными SQL и его реализация.

В материалах рассматриваются основные понятия из области баз данных, описываются функции банка данных и СУБД, свойства данных, поддерживаемые в базе, используемые языки, уровни представления данных.

Лекция №1. Информация и данные

Каждой цивилизации приходилось иметь дело с обработкой информации. С развитием экономики и ростом численности населения возрастает и объем взаимосвязанных данных, необходимых для решения экономических, административных и управленческих задач. Выделим 3 области, о которых можно говорить при обсуждении понятия «информация» (рисунок 1).

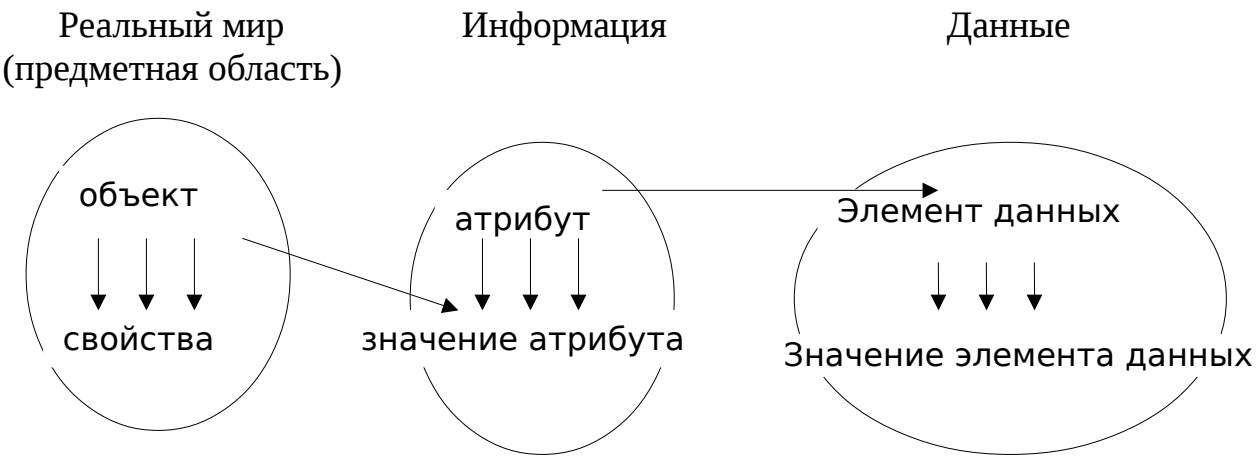


Рисунок 1 - Информация и данные

Первая область – это реальный мир, в котором существует множество объектов, обладающих определенными свойствами. В реальном мире выделяют некоторую предметную область, которая является частью реального мира. Примерами предметной области могут служить наука, животные, книга, автомашина, компьютер, язык программирования, изучаемая дисциплина, техническая литература, сессия, живопись, кино, спорт, флора, фауна, транспорт, связь, больница, аэропорт, институт, театр и т.д.

В предметной области можно выделить один или несколько объектов. Объектом может быть человек, предмет, событие, место, явление, понятие. Например, для предметной области «Больница» можно выделить такие объекты, как «Больница», «Больной», «Врач», «Палата», «Отделение», «Младший медперсонал», «Лаборатория», «Лекарства» и т.д. Каждый объект должен характеризоваться несколькими свойствами. Например, объект «Больница» может описываться с помощью таких свойств, как «Номер больницы», «Наименование больницы», «Фамилия главного врача», «Количество койко-мест», «Количество персонала», «Фонд заработной платы» и т.д. Объект «Больной» может быть описан с помощью таких свойств как «Фамилия и инициалы больного», «Номер лечебной карточки», «Номер палаты», «Диагноз», «Лечение», «Фамилия лечащего врача» и т.д. Точность, полнота и объемность описания как предметной области, так и отдельных

объектов зависит от решаемой задачи и потребностей пользователей в информации.

Вторая область – область информации, существующей в представлении людей. Здесь говорят об атрибутах объектов и обозначают атрибуты символически на естественном языке или на языке программирования или в какой-либо другой форме. Атрибутам приписывают значения. Например, для атрибута «Фамилия и инициалы больного» объекта «Больной» можно ввести такие обозначения: « ФИО», «ФИО больного», «FIO», «bolnoy». Приведем примеры значений этого атрибута: «Каримов А.К», «Иванов И.А» и т. д.

Третья область – это область данных, хранящихся в компьютере. Элемент данных представляет собой действительные данные, соответствующие определенному атрибуту, связанному с конкретным объектом из предметной области. Значения элементов данных в памяти компьютера представляют собой строки символов или битов. В зависимости от того, как элементы данных описывают объект, они могут быть количественными, качественными или описательными. Значения элементов данных могут существовать независимо от информации, которая запоминается с их помощью. Но смысл они приобретут только тогда, когда будут привязаны к определенному элементу. Например, в компьютере постоянно хранятся такие значения: «красный», « синий», «зеленый», «желтый». Затем их можно связать с конкретными элементами данных: «красный мак», « синее небо», «зеленая трава», «желтый светофор». В таблице 1 приведено описание объекта «Больница».

Таблица 1 - Описание объекта «Больница»

Объект	Атрибут	Значение атрибута
больница	Номер больницы	5
	Наименование больницы	ЦГКБ
	Фамилия главного врача	Каримов
	Количество койко-мест	230
	Количество персонала	150
	Фонд заработной платы	400000

Соответственно двум понятиям «информация» и «данные» в базах данных различают 2 аспекта рассмотрения вопросов:

- инфологический аспект употребляется при рассмотрении вопросов, связанных со смысловым содержанием данных независимо от способов их представления в памяти компьютера;
- датологический аспект употребляется при рассмотрении вопросов представления данных в памяти компьютера.

Информационные отношения и взаимосвязи данных.

Между различными объектами, свойствами объектов и объектами и их свойствами существуют объективные, определяемые предметной областью отношения. При информационном отображении объектов и присущих им свойств эти отношения переносятся и на элементы информации. Отношения между элементами информации информационны по своей природе, отображая реально существующие взаимосвязи, и многообразны по своему характеру, внутреннему механизму, математической интерпретации.

Даже для простейшего случая *r(z, s)* , где *z* и *s* – простые переменные, *r* – отношение между ними, это отношение является одним из бесконечного множества отношений, определяемых как типом переменных *z* и *s*, так и многими другими обстоятельствами. Так что для данного случая точнее говорить об отношении $r_i(z,s)$, выбранном из множества возможных отношений: $R(r_1,r_2,...,r_i,...)$. Рассмотрим пример: пусть *z* = 3, а *s* = 6. Тогда можно определить такие отношения, существующие между *z* и *s*:

1) *z* и *s* – числа.
2) *z* и *s* - целые числа.
3) *z* и *s* - натуральные числа.
4) *z* и *s* - положительные числа.
5) *z* меньше *s*.
6) *z* = *s* – 3.
7) *s* = *z**2.
8) *s* = *z*/2.

Можно определить и другие отношения.

Среди множества отношений могут быть отношения следующих видов:

- теоретико-множественного характера – принадлежность какому-то множеству, иерархическое отношение, отношение эквивалентности элементов и др.;
- логической природы – предикатные отношения, отношения операндов логического выражения или аргументов логической функции и др.;
- арифметического типа – сравнение чисел, упорядоченность чисел, функциональная зависимость и др;
- лексикографической упорядоченности – упорядоченность информации символьного типа в соответствии с алфавитом используемого языка (например, список фамилий студентов группы).

Имеются и другие отношения самого разнообразного типа, природа которых подчас не формализуема или требует для формализации слишком сложного аппарата. Для рассмотренного выше примера отношения 1 – 4 носят теоретико-множественный характер и указывают принадлежность переменных z и s одному из множеств. Отношения 5 – 8 относятся к арифметическому типу и указывают сравнение переменных z и s и функциональные зависимости между ними. Приведем примеры бинарных отношений:

1) Астана – столица Казахстана.

2) Канат Ахметов имеет профессию инженера – системотехника.

3) факультет аэрокосмических и информационных технологий является одним из факультетов АУЭС.

Взаимосвязь выражает отображение или связь между двумя множествами данных. Различают взаимосвязи типа «один к одному» (1:1), «один ко многим» (1:M или 1: ∞), «многие ко многим» (M : M).

Пусть имеется объект «СТУДЕНТ». При его описании используем атрибуты «ФАМИЛИЯ», «ИМЯ», «ГОД РОЖДЕНИЯ», «НОМЕР ЗАЧЕТНОЙ КНИЖКИ», «НОМЕР ГРУППЫ». Для этого примера считаем, что фамилии у студентов не повторяются, т.е. нет однофамильцев. Между атрибутами имеются следующие взаимосвязи:

1) «Один к одному»

ФАМИЛИЯ → НОМЕР ЗАЧЕТКИ

2) «Один ко многим»

НОМЕР ГРУППЫ →→ НОМЕР ЗАЧЕТКИ

3) «Многие ко многим»

ГОД РОЖДЕНИЯ →→ НОМЕР ГРУППЫ

Аналогично такие взаимосвязи могут быть установлены и между объектами.

Пусть имеются объект «СТУДЕНТ» с указанными выше атрибутами и объект «ГРУППА» с атрибутами «НОМЕР ГРУППЫ», «КОЛИЧЕСТВО СТУДЕНТОВ», «ФАМИЛИЯ СТАРОСТЫ». Между указанными объектами «ГРУППА» и «СТУДЕНТ» существует взаимосвязь «Один ко многим»:

ГРУППА →→ СТУДЕНТ

а между объектами СТУДЕНТ и ГРУППА существует взаимосвязь «Один к одному»:

СТУДЕНТ → ГРУППА

Контрольные вопросы:

1) Как рассматривается понятие «информация»?

2) Что такое предметная область реального мира?

3) Как описывается объект предметной области?

4) Что такое информационные отношения?

5) Какие виды взаимосвязей существуют между объектами?

Лекция №2. Системы баз данных

База данных (БД) представляет собой поименованную совокупность данных, отображающую состояние множества объектов из рассматриваемой предметной области, их атрибутов (свойств) и взаимоотношений. Практически база данных является информационной моделью предметной области, от обоснованности, точности и достоверности которой зависит эффективность информационной системы. Запись данных представляет собой совокупность

значений атрибутов, описывающих конкретный объект реального мира. Первичный ключ – это атрибут (группа атрибутов), идентифицирующий объект уникальным образом. Вторичный ключ идентифицирует группу записей.

Под банком данных (БнД) понимается организационно-техническая система, представляющая собой совокупность баз данных, технических и программных средств формирования и ведения этих баз и коллектива специалистов, обеспечивающих функционирование этой системы. Фактически банк данных есть совокупность базы данных и системы управления базой данных (СУБД). Такая структура банка данных представлена на рисунке 2.

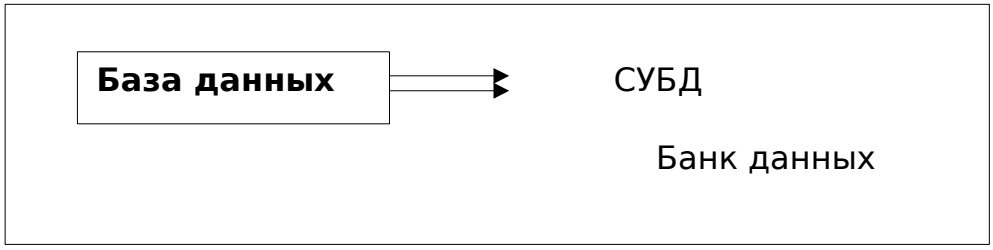


Рисунок 2 - Упрощенная структура банка данных

Отличительными чертами банка данных являются:

- удовлетворение актуальных информационных потребностей множества пользователей, проведение поиска информации по произвольным запросам, выдача информации пользователю в различных формах;
- обеспечение возможности хранения и модификации больших объемов многоаспектной информации, обеспечение доступа к данным только пользователей с соответствующими полномочиями;
- обеспечение требуемого уровня достоверности хранимой информации и ее непротиворечивости;
- возможность реорганизации и расширения баз данных при изменении границ предметной области;
- обеспечение заданных требований эффективности функционирования.

Преимущества банка данных: сокращение избыточности данных, устранение противоречивости хранимых данных, многоаспектное использование данных, комплексная оптимизация при проектировании структур баз данных, обеспечение возможности санкционированного доступа к данным. К недостаткам банка данных можно отнести сложность создаваемых систем, их чувствительность к последствиям сбоев и аварий. Структура банка данных с учетом всех его компонентов представлена на рисунке 3.

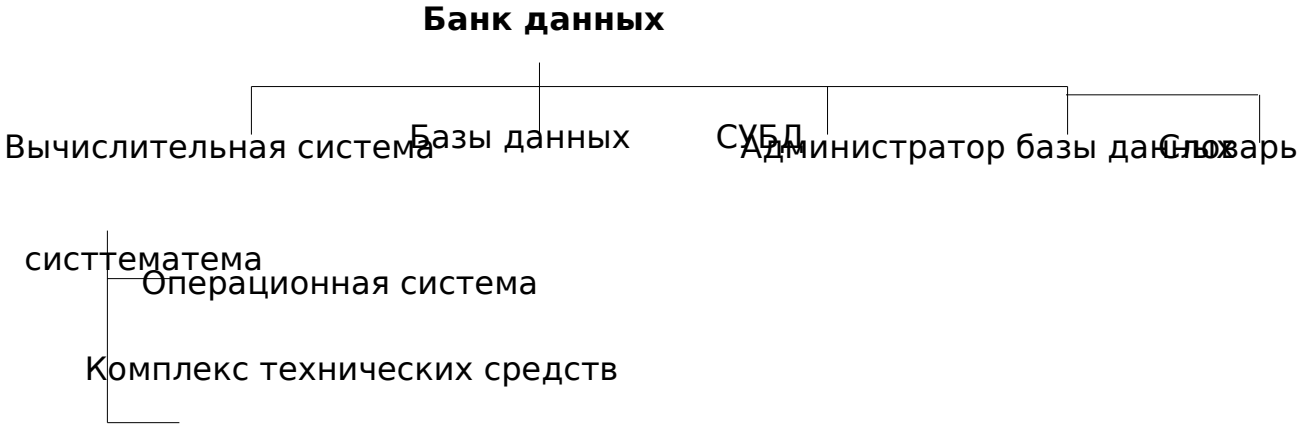


Рисунок 3 - Структура банка данных

Словарь хранит информацию обо всех ресурсах: об объектах, их свойствах и отношениях для каждой предметной области, о возможных значениях и форматах представления данных, об источниках возникновения данных, о кодах защиты и разграничениях доступа, о наименовании данных, смысловом описании, структуре, связях и др.

Системы управления базами данных.

Для создания и использования БД служат системы управления базами данных (СУБД), которые занимают особую позицию в мире программного обеспечения и нашей повседневной жизни. Системы управления базами данных обеспечивают реализацию новых концепций в организации информационных служб через создание информационных систем на основе технологии баз данных. В настоящее время широко применяются муниципальные, банковские, биржевые информационные системы, информационные системы оптовой и розничной торговли, торговых домов, служб управления трудом и занятостью, базы данных рынка товаров и услуг, справочной и аналитико-прогнозной котировочной информации и др. Как правило, работа этих систем осуществляется в локальных вычислительных сетях различной архитектуры или их объединениях, получивших название корпоративных сетей, дальнейшая интеграция которых возможна с помощью глобальной сети Интернет.

Грамотная организация информационно-вычислительной системы, в том числе и с использованием Интернет, позволяет не только экономить труд и время специалистов, но вместе с тем увеличивать творческую долю труда работников различных категорий, использующих данную систему в своей работе. Другими словами, использование такой системы, в первую очередь, увеличивает качество труда, а в некоторых случаях позволяет добиться прямого экономического эффекта. Технология систем баз данных позволяет с

наименьшими затратами и наибольшей эффективностью организовать информационную систему для решения пользовательских задач и является одной из перспективных технологий обработки данных.

Данные – это информация, зафиксированная в определённой форме, пригодной для последующей обработки, хранения и передачи. Данные соответствуют зарегистрированным фактам об объектах или явлениях реального мира.

В традиционной технологии обработки данных каждая программа работает со своими файлами данных, что приводит к дублированию данных. Попытки избежать дублирования данных за счет использования одних и тех же файлов данных в различных программах приводят к зависимости программ друг от друга. Дело в том, что в системах традиционной обработки данных форматы хранимых данных и структура файла данных определяются программой его создающей и должны учитываться в программах, использующих этот файл. Изменения в структуре файла данных приводят к изменениям в использующих его программах.

Чтобы избежать дублирования данных и зависимости между данными и программами, необходимо иметь файлы данных, создание и изменение структуры которых не определяется программой какого-либо конкретного пользователя. Совокупность таких файлов стали называть базой данных. Программные системы, которые были созданы для разработки и управления базами данных, получили название систем управления базами данных.

База данных (БД) — это специальным образом организованные массивы данных, хранящиеся в вычислительной системе и независимые от использующих их программ (именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области). Под предметной областью (ПО) понимается часть реального мира, интересующая пользователя, – это область применения конкретной БД. Различают БД, применяемые в сфере управления предприятиями и организациями, транспортом, в медицине, научных исследованиях и т.д.

Система управления базами данных (СУБД) — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Система баз данных (СБД) - это система специальным образом организованных данных - баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

История развития СУБД насчитывает более 40 лет. В 1968 году была введена в эксплуатацию первая промышленная СУБД система IMS фирмы IBM. В 1975 году появился первый стандарт ассоциации по языкам систем обработки данных — Conference of Data System Languages (CODASYL), который определил ряд фундаментальных понятий в теории систем баз

данных, которые и до сих пор являются основополагающими для сетевой модели данных.

В дальнейшее развитие теории баз данных большой вклад был сделан американским математиком Э. Ф. Коддом, который является создателем реляционной модели данных. В 1981 году Э. Ф. Кодд получил за создание реляционной модели и реляционной алгебры престижную премию Тьюринга Американской ассоциации по вычислительной технике.

Первый этап развития СУБД связан с организацией баз данных на больших машинах типа IBM 360/370, ЕС-ЭВМ и мини-ЭВМ типа PDP11 (фирмы Digital Equipment Corporation — DEC), разных моделях HP (фирмы Hewlett Packard).

Базы данных хранились во внешней памяти центральной ЭВМ, пользователями этих баз данных были задачи, запускаемые, в основном, в пакетном режиме.

Все СУБД базируются на мощных мультипрограммных операционных системах (MVS, SVM, RTE, OSRV, RSX, UNIX), поэтому, в основном, поддерживается работа с централизованной базой данных в режиме распределенного доступа.

Функции управления распределением ресурсов в основном осуществляются операционной системой (ОС). Поддерживаются языки низкого уровня манипулирования данными, ориентированные на навигационные методы доступа к данным. Значительная роль отводится администрированию данных.

С появлением персональных компьютеров связан следующий этап развития баз данных. Особенности этого этапа следующие.

Все СУБД были рассчитаны на создание БД в основном с монопольным доступом. Большинство СУБД имели развитый и удобный пользовательский интерфейс. В большинстве существовал интерактивный режим работы с БД как в рамках описания БД, так и в рамках проектирования запросов. Кроме того, большинство СУБД предлагали развитый и удобный инструментарий для разработки готовых приложений без программирования. Во всех настольных СУБД поддерживался только внешний уровень представления реляционной модели, то есть только внешний табличный вид структур данных.

При наличии высокоуровневых языков манипулирования данными типа реляционной алгебры и SQL в настольных СУБД поддерживались низкоуровневые языки манипулирования данными на уровне отдельных строк таблиц.

Наличие монопольного режима работы фактически привело к вырождению функций администрирования БД и в связи с этим - к отсутствию инструментальных средств администрирования БД.

И, наконец, последняя и в настоящий момент весьма положительная особенность - это сравнительно скромные требования к аппаратному обеспечению со стороны настольных СУБД.

Следующий этап развития баз данных - это распределенные базы данных и клиент-серверная архитектура.

Практически все современные СУБД имеют средства подключения клиентских приложений, разработанных с использованием настольных СУБД, и средства экспорта данных из форматов настольных СУБД.

Прикладные программы (ПП) пользователей обращаются к СУБД на логическом уровне без учета того, как данные хранятся в БД. СУБД по запросу пользователя определяет необходимые физические файлы, посредством обращения к описанию отображения данных, и порядок доступа к ним. После чтения данных информация приводится к необходимому прикладной программе виду.

От других способов организации данных СБД отличает ряд существенных преимуществ, таких как:

- значительное сокращение избыточности информации;
- независимость данных от программ и программ от данных;
- качественное управление данными;
- обеспечение контроля на целостность и непротиворечивость данных;
- уменьшение затрат на хранение и обработку данных.

Технология баз данных имеет не только достоинства. Каждое обращение к БД осуществляется при помощи описания отображения данных, что определяет достоинства и недостатки этой технологии:

- увеличение доли служебной информации в общем объеме хранимых данных;
- повышенные требования к техническим и программным средствам системы, так как часть ресурсов расходуется на нужды самой системы;
- потеря эффективности отдельных приложений;
- последствия сбоев труднее исправлять по сравнению с традиционной технологией обработки данных.

Организация хранения и обработки данных - важнейший вопрос, стоящий перед разработчиками информационных систем. При разработке и эксплуатации систем, основанных на технологии баз данных, необходимо учитывать ряд требований, которым они должны удовлетворять:

- адекватность отображения предметной области;
- организация взаимодействия с системой пользователей разного уровня и в различных режимах;
- обеспечение секретности, надёжности и достоверности данных;
- независимость данных от программ их использующих и программ от данных;
- обеспечение приемлемых характеристик функционирования (время реализации запроса, требуемый объём памяти, сервис, стоимость системы и т.д.).

Система управления базой данных представляет собой совокупность языковых и программных средств, предназначенных для создания и ведения

баз данных. Централизованное управление базами данных посредством СУБД обеспечивает:

- сокращение избыточности и устранение несовместимости в хранимых данных;
- совместное использование данных множеством пользователей и приложений, что достигается необходимой интеграцией данных
- стандартизация представления данных, упрощающая эксплуатацию банка данных;
- разграничение доступа к данным;
- целостность данных, которая достигается за счет процедур, предотвращающих включение в базу неверных данных, и восстановление базы данных после аварий и сбоев.

СУБД должна поддерживать следующие функции:

- ввод данных, их контроль и печать выявленных ошибок;
- обновление и корректировка данных;
- размещение, формирование и ведение файлов баз данных;
- обработка запросов пользователей и приложений;
- контроль и диагностика состояния банка данных;
- восстановление базы данных в случае аварии;
- реорганизация и реструктуризация баз данных;
- создание и поддержка библиотеки типовых процедур, функций и макрокоманд.

Обработка запросов в банке данных.

Процесс обработки запроса на данные с помощью СУБД представлен на рисунке 4. Рассмотрим этот процесс. Рабочая область приложения (РОП) представляет собой область памяти, в которой находятся данные, предназначенные для взаимодействия между приложением и СУБД. Фактически РОП является зоной загрузки и разгрузки данных, совместно используемых приложением и СУБД. Системный буфер также является областью памяти, совместно используемой операционной системой и СУБД. Как правило, СУБД имеет несколько системных буферов.

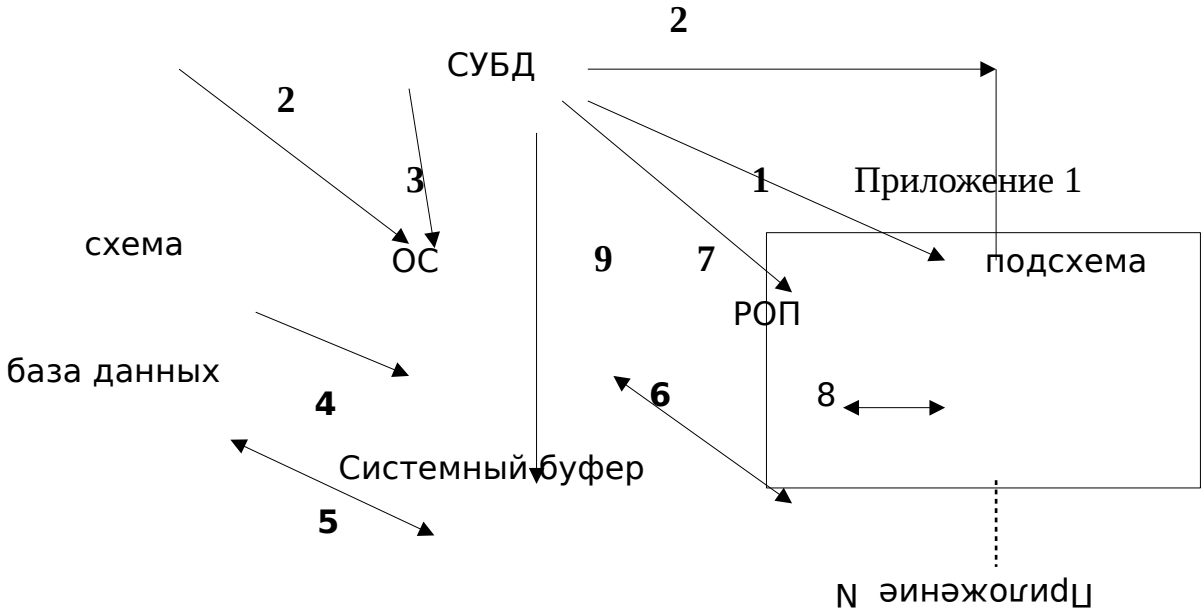


Рисунок 4 - Обработка запроса от приложения

Опишем назначение связей:

1 – приложение обращается к СУБД за данными, хранящимися в БД, с помощью операторов ЯМД;

2 – СУБД анализирует запрос на данные и дополняет его информацией из схемы и подсхемы, определяя, где хранятся требуемые данные и в каком виде они должны быть предоставлены приложению;

3 – СУБД вызывает выполнение операционной системой (ОС) физических операций ввода / вывода, необходимых для выполнения запроса;

4 – ОС взаимодействует с внешней памятью, в которой хранится база данных;

5 – данные передаются из БД в системный буфер;

6 – СУБД передает данные из системного буфера в РОП приложения, при этом СУБД выполняет все необходимые преобразования данных из их представления в БД, определяемого схемой, в их представление для приложения, определяемое подсхемой;

7 – в ходе обработки запроса СУБД передает в приложение информацию о найденных ошибках или о нормальном ходе работы;

8 – данные из РОП может использовать приложение;

9 – СУБД управляет системными буферами, которые используются при обработке запросов от всех приложений.

Аналогично обрабатываются запросы от всех приложений, которые обращаются к СУБД за данными, хранящимися в базе данных.

Контрольные вопросы:

1) Что такое база данных?

2) Что включает в свой состав банк данных?
3) Каковы преимущества банка данных?
4) Какие функции поддерживает СУБД?
5) Как проходит процесс обработки запроса к СУБД от приложения?

Лекция №3. Свойства данных, поддерживаемые в базе

В БД поддерживаются такие свойства данных, как интеграция, независимость, отсутствие дублирования, защита и целостность. База данных позволяет осуществить интеграцию данных. Это означает, что БД содержит данные для множества приложений и пользователей, каждый из которых работает лишь с небольшой частью данных, хранящихся в БД. Отдельные элементы данных могут обрабатываться и использоваться в нескольких приложениях.

В базе данных необходимо обеспечить независимость приложений от данных по двум причинам:

- в разных приложениях одни и те же данные необходимо представлять по-разному;

- в БД должна иметься возможность изменять структуру хранения или стратегию доступа к данным так, чтобы в случае изменения требований не надо было модифицировать приложения.

В связи с этим говорят о логической и физической независимости данных. Логическая независимость данных означает, что общая логическая структура – схема данных – может быть изменена без изменения всех приложений. Физическая независимость данных означает, что физическое расположение и организация данных во внешней памяти компьютера могут изменяться, не вызывая при этом изменений ни схемы данных, ни приложений.

В БД должно обеспечиваться отсутствие дублирования или избыточности данных. Фактически БД можно определить как неизбыточную совокупность данных. Однако при обработке для уменьшения времени доступа или упрощения адресации и поиска данных во многих БД избыточность присутствует в некоторой степени. Приходится идти на компромисс для обеспечения более быстрой обработки или восстановления данных в случае сбоев. Поэтому говорят об управляемом или минимальном уровне дублирования данных.

Защита данных означает, во-первых, предупреждение несанкционированного или случайного доступа к данным, их изменения или разрушения со стороны пользователей, во-вторых, предупреждение изменения или разрушения данных при сбоях технических или программных средств и ошибках в работе администратора БД. Защита данных обеспечивает их безопасность и секретность. Под функцией безопасности понимается защита данных от непреднамеренного доступа к данным и от возможности их искажения со стороны пользователей или лиц, выполняющих эксплуатацию

банка данных, а также при сбоях в технических и программных средствах. Обеспечение безопасности – внутренняя задача банка данных. Под функцией секретности понимается защита данных от преднамеренного доступа к данным пользователей или лиц, выполняющих эксплуатацию БнД, или посторонних лиц с целью получения секретной информации или преднамеренного изменения и искажения данных. Как правило, в БД данные делятся на общедоступные и секретные. Обеспечение секретности – внешняя задача банка данных, решение вопросов секретности находится в компетенции юридических и административных органов БнД.

Целостность данных означает безошибочность, точность и достоверность данных в БД в каждый момент времени. Целостность обеспечивается набором специальных правил, устанавливающих допустимость данных и связей между ними, называемых ограничениями целостности. Ограничения целостности могут относиться к различным объектам БД: атрибутам, записям, отношениям, связям между ними и т.п. Например, для атрибутов могут быть установлены следующие ограничения целостности:

- заданный тип и формат атрибута автоматически допускают ввод данных только указанного типа; например, если атрибут имеет тип «Дата» в формате ДД.ММ.ГГ (день – месяц – год), то число дней не может превышать 31, а число месяцев – 12;
- задание диапазона значений, как правило, используется для числовых полей; например, стоимость товара должна быть > 200 или стоимость товара должна быть >100 и <500;
- недопустимость пустого значения для обязательных атрибутов; например, обязательно должен быть задан номер автомобиля или фамилия сотрудника;
- задание списка значений используется, как правило, для текстовых (символьных) атрибутов; например, наименование должности сотрудника лучше выбирать из заранее созданного списка.

Для реализации ограничений целостности, имеющих отношение к записям, таблицам или связям между ними, в СУБД используются триггеры. Триггер – это предварительно определенное действие или последовательность действий автоматически осуществляемых при выполнении операций обновления, добавления или удаления данных. Триггер является мощным инструментом контроля за изменением данных в БД и помогает программисту автоматизировать операции, которые должны выполняться в этом случае. Триггер включает в себя следующие компоненты:

- ограничения, для реализации которых и создается триггер;
- событие, которое характеризует возникновение ситуации, требующей проверки ограничений целостности;
- действие, выполняемое за счет одной или нескольких процедур, в которых заложена логика реализации ограничений целостности.

Схема и подсхема.

Схема представляет собой логическое описание всех хранимых данных. Схема содержит имена объектов, их атрибутов и взаимосвязей между ними. Подсхема определяет собой описание данных, которые используются каким-либо приложением или пользователем. На основе одной схемы можно построить множество подсхем.

Пример схемы данных для информационной системы « Автомобиль», включающий три таблицы: «Авто», «Завод», «ВладелецАвто», приведён на рисунке 5.

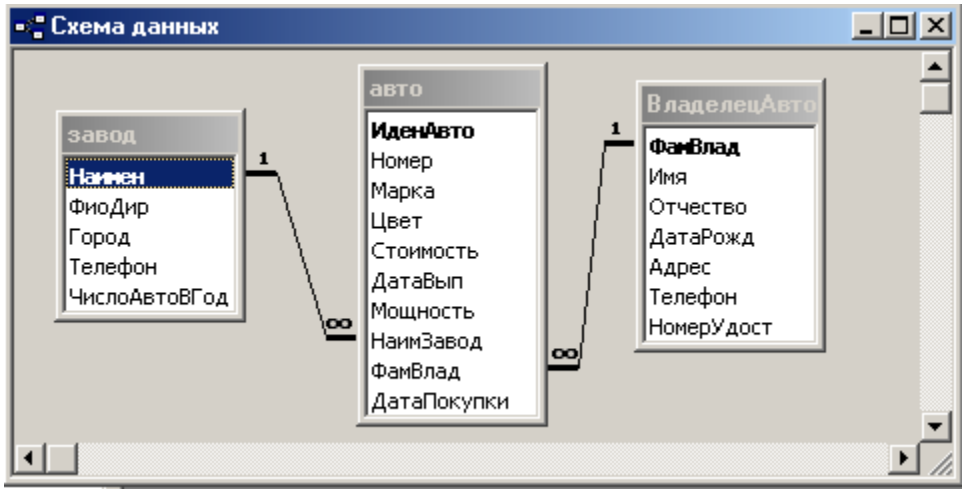


Рисунок 5 – Схема данных

Пользователи банка данных.

С базой данных взаимодействуют несколько типов пользователей: проектировщик БД, прикладной программист, конечный пользователь (непрограммист), администратор базы данных (АБД). Проектировщик БД решает все вопросы, связанные с созданием БД, выбором и использованием СУБД. Кроме того, в ходе эксплуатации БД проектировщик может принимать участие в решении вопросов по дальнейшему совершенствованию БД. Прикладной программист разрабатывает приложения (прикладные программы), использующие данные из БД и предназначенные для пользователей БД. Прикладной программист создает подсхему для разрабатываемого приложения и пишет запросы к СУБД на языке манипулирования данными в соответствии с принятыми правилами и нормами. При этом он работает в тесном контакте с администратором БД. Конечный пользователь обращается в банк данных с запросами на поиск требуемых данных. Эти пользователи имеют разный уровень профессиональной подготовки и, как правило, не являются профессионалами в области информационных технологий. Они используют язык запросов пользователей, который является языком, близким к естественному языку для рассматриваемой предметной области.

Администратор базы данных отвечает за создание и ведение БД. АБД включает в свой состав группу сопровождения, группу контроля функционирования банка данных, эксперта по языкам запросов, эксперта по разработке приложений, эксперта по системным вопросам, эксперта по вопросам эксплуатации. В функции администратора БД входят:

- координация всех действий по проектированию, реализации и ведению БД, учет перспективных и текущих требований всех пользователей с целью удовлетворения их актуальных информационных потребностей;
- разработка и реализация мер по обеспечению защиты данных от некомпетентного их использования, от сбоев технических и программных средств;
- разграничение доступа к данным и обеспечение их секретности;
- контроль за избыточностью и противоречивостью данных, обеспечение их достоверности;
- проведение при необходимости реорганизации и реструктуризации данных;
- координация работы системных и прикладных программистов.

Языки, используемые в банке данных.

В банке данных используется 4 типа языков: язык описания данных (ЯОД), язык манипулирования данными (ЯМД), язык запросов пользователя (ЯЗП) и базовый язык (БЯ).

Язык описания данных является языком декларативного типа, используется для логического описания данных и поддерживает функции, позволяющие присваивать уникальные имена каждому элементу данных, идентифицировать типы элементов данных (поле, сегмент, запись, набор, отношение, атрибут, таблица и др.), специфицировать порядок вхождения одних элементов данных в другие, устанавливать существующие взаимосвязи между данными.

Язык манипулирования данными позволяет реализовать интерфейс между приложением и СУБД, поддерживает такие функции, как открыть / закрыть файл базы данных, найти требуемые элементы данных, изменить / добавить / удалить некоторые данные и т.д. Фактически ЯМД выполняет все функции по обработке данных: загрузка, корректировка, поиск.

Язык запросов пользователя позволяет выбирать из базы все требуемые пользователю данные в соответствии с задаваемыми им критериями, этот язык должен быть близок к естественному языку для рассматриваемой предметной области.

Базовый язык представляет собой универсальный язык программирования, в среде которого реализована СУБД (например, язык С++).

Контрольные вопросы:

1) Что такое интеграция данных?

2) Какие виды независимости данных поддерживаются в базе?

3) Что такое целостность данных?
4) Что такое схема данных?
5) Что такое подсхема?
6) Какие группы пользователей имеются в банке данных?
7) Какие функции выполняет администратор базы данных?
8) Что такое ЯОД?
9) Что такое ЯМД?
10) Для каких целей нужен язык запросов?

Лекция №4. Архитектура систем баз данных

Основные концепции. Независимость данных. Основные компоненты данных.

Активная деятельность по отысканию приемлемых способов обобществления непрерывно растущего объема информации привела к созданию в начале 60-х годов специальных программных комплексов, называемых «*Системы управления базами данных*» (СУБД). Основная особенность СУБД – это наличие процедур для ввода и хранения не только самих данных, но и описаний их структуры. Файлы, снабженные описанием хранимых в них данных и находящиеся под управлением СУБД, стали называть банки данных, а затем «*Базы данных*» (БД). Пусть, например, требуется хранить расписание движения самолетов (рисунок 1.) и ряд других данных, связанных с организацией работы аэропорта (БД «Аэропорт»). Используя для этого одну из современных «русифицированных» СУБД, можно подготовить следующее описание расписания.

```
СОЗДАТЬ ТАБЛИЦУ Расписание
 (Номер_рейса        Целое
  Дни_недели         Текст (8)
  Пункт_отправления  Текст (24)
  Время_вылета       Время
  Пункт_назначения   Текст (24)
  Время_прибытия     Время
  Тип_самолета       Текст (8)
  Стоимость_билета   Валюта);
```

И ввести его вместе с данными в БД «Аэропорт».

Язык запросов СУБД позволяет обращаться за данными как из программ, так и с терминалов, сформировав запрос.

```
ВЫБРАТЬ  Номер_рейса, Дни_недели, Время_вылета
```

```
ИЗ ТАБЛИЦЫ  Расписание
ГДЕ  Пункт_отправления = 'Москва'
  И  Пункт_назначения  = 'Киев'
  И  Время_вылета > 17;
```

Получим расписание "Москва-Киев" на вечернее время, а по запросу.

```
ВЫБРАТЬ  КОЛИЧЕСТВО(Номер_рейса)
ИЗ ТАБЛИЦЫ  Расписание
ГДЕ  Пункт_отправления = 'Москва'
  И  Пункт_назначения  = 'Минск';
```

Получим количество рейсов "Москва-Минск".

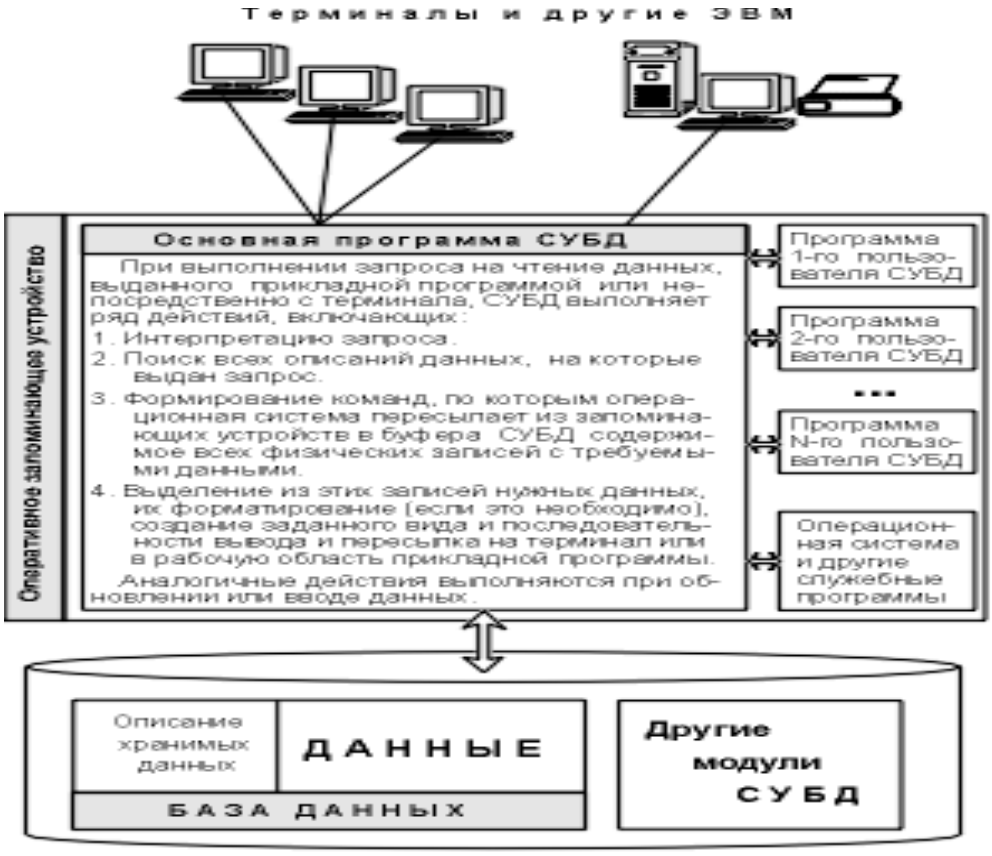


Рисунок 6 - Связь программ и данных при использовании СУБД

Эти запросы не потеряют актуальности и при расширении таблицы.

```
ДОБАВИТЬ В ТАБЛИЦУ Расписание
  Длительность_полета  Целое;
```

Как это было с программами обработки почтовых адресов при введении почтового индекса.

Однако, на обмен данными через СУБД требуется большее время, чем на обмен аналогичными данными прямо из файлов, специально созданных для того или иного приложения.

Уровни архитектуры баз данных.

СУБД должна предоставлять доступ к данным любым пользователям, включая и тех, которые практически не имеют и (или) не хотят иметь представления о:

- физическом размещении в памяти данных и их описаний;
- механизмах поиска запрашиваемых данных;
- проблемах, возникающих при одновременном запросе одних и тех же данных многими пользователями (прикладными программами);
- способах обеспечения защиты данных от некорректных обновлений и (или) несанкционированного доступа;
- поддержании баз данных в актуальном состоянии и множестве других функций СУБД.

При выполнении основных из этих функций СУБД должна использовать различные описания данных. А как создавать эти описания? Естественно, что проект базы данных надо начинать с анализа предметной области и выявления требований к ней отдельных пользователей (сотрудников организации, для которых создается база данных). Подробнее этот процесс будет рассмотрен ниже, а здесь отметим, что проектирование обычно поручается человеку (группе лиц) – *администратору базы данных* (АБД). Им может быть как специально выделенный сотрудник организации, так и будущий пользователь базы данных, достаточно хорошо знакомый с машинной обработкой данных. Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, АБД сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных, называют *инфологической моделью данных* (рисунок 7).

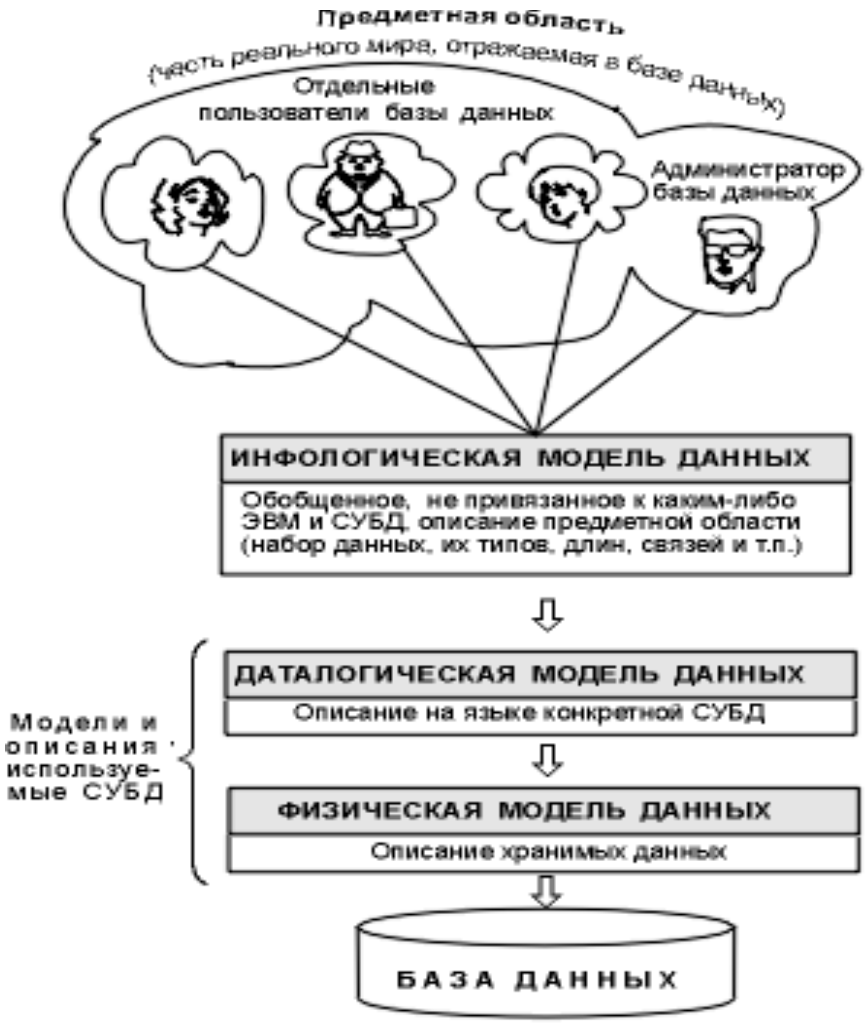


Рисунок 7 - Уровни моделей данных

Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных. В конце концов этой средой может быть память человека, а не ЭВМ. Поэтому инфологическая модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторого определения, чтобы эта модель продолжала отражать предметную область. Остальные модели, показанные на рисунке 7, являются компьютеро - ориентированными. С их помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных. Нужные данные отыскиваются СУБД на внешних запоминающих устройствах по *физической модели данных*. Так как указанный доступ осуществляется с помощью конкретной СУБД, то модели должны быть описаны на *языке описания данных* этой СУБД. Такое описание, создаваемое АБД по инфологической модели данных, называют *даталогической моделью данных*. Трехуровневая архитектура (инфологический, даталогический и физический уровни) позволяет обеспечить *независимость хранимых данных* от использующих их программ. АБД может при необходимости переписать хранимые данные на другие носители информации и (или) реорганизовать их

физическую структуру, изменив лишь физическую модель данных. АБД может подключить к системе любое число новых пользователей (новых приложений), дополнив, если надо, даталогическую модель. Указанные изменения физической и даталогической моделей не будут замечены существующими пользователями системы (окажутся «прозрачными» для них), так же как не будут замечены и новые пользователи. Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.

Контрольные вопросы:

1) Как вы представляйте архитектуру базы данных?
2) Что такое уровни базы данных?
3) Что такое СУБД?
4) Сущность инфологического подхода?
5) Сколько уровней имеет стандартная архитектура баз данных?
6) Что собой представляет внешняя модель данных?
7) Что собой представляет концептуальная модель данных?
8) Что собой представляет внутренняя модель данных?
9) Как называется уровень архитектуры баз данных, доступный пользователям любой квалификации?

Лекция №5. Проектирование баз данных

Концептуальная модель предметной области. Логические модели.

Только небольшие организации могут обобществить данные в одной полностью интегрированной базе данных. Чаще всего администратор баз данных (даже если это группа лиц) практически не в состоянии охватить и осмыслить все информационные требования сотрудников организации (т.е. будущих пользователей системы). Поэтому информационные системы больших организаций содержат несколько десятков БД, нередко распределенных между несколькими взаимосвязанными ЭВМ различных подразделений (так в больших городах создается не одна, а несколько овощных баз, расположенных в разных районах). Отдельные БД могут объединять все данные, необходимые для решения одной или нескольких прикладных задач, или данные, относящиеся к какой-либо предметной области (например, финансам, студентам, преподавателям, кулинарии и т.п.). Первые обычно называют *прикладными БД*, а вторые – *предметными БД* (соотносящимся с предметами организации, а не с ее информационными приложениями), (первые можно сравнить с базами материально-технического снабжения или отдыха, а вторые – с овощными и обувными базами). Предметные БД позволяют обеспечить поддержку любых текущих и будущих приложений, поскольку набор их элементов данных включает в себя наборы элементов данных прикладных БД. Вследствие этого предметные БД создают основу для обработки неформализованных, изменяющихся и неизвестных

запросов и приложений (приложений, для которых невозможно заранее определить требования к данным). Такая гибкость и приспосабливаемость позволяет создавать на основе предметных БД достаточно стабильные информационные системы, т.е. системы, в которых большинство изменений можно осуществить без вынужденного переписывания старых приложений. Основывая же проектирование БД на текущих и предвидимых приложениях, можно существенно ускорить создание высокоэффективной информационной системы, т.е. системы, структура которой учитывает наиболее часто встречающиеся пути доступа к данным. Поэтому прикладное проектирование до сих пор привлекает некоторых разработчиков. Однако по мере роста числа приложений таких информационных систем быстро увеличивается число прикладных БД, резко возрастает уровень дублирования данных и повышается стоимость их ведения. Таким образом, каждый из рассмотренных подходов к проектированию воздействует на результаты проектирования в разных направлениях. Желание достичь и гибкости, и эффективности привело к формированию методологии проектирования, использующей как предметный, так и прикладной подходы. В общем случае предметный подход используется для построения первоначальной информационной структуры, а прикладной – для ее совершенствования с целью повышения эффективности обработки данных. При проектировании информационной системы необходимо провести анализ целей этой системы и выявить требования к ней отдельных пользователей (сотрудников организации). Сбор данных начинается с изучения сущностей организации и процессов, использующих эти сущности (подробнее в приложении Б). Сущности группируются по «сходству» (частоте их использования для выполнения тех или иных действий) и по количеству ассоциативных связей между ними (самолет – пассажир, преподаватель – дисциплина, студент – сессия и т.д.). Сущности или группы сущностей, обладающие наибольшим сходством и (или) с наибольшей частотой ассоциативных связей объединяются в предметные БД. (Нередко сущности объединяются в предметные БД без использования формальных методик – по «здравому смыслу».) Для проектирования и ведения каждой предметной БД (нескольких БД) назначается АБД, который далее занимается детальным проектированием базы. Далее будут рассматриваться вопросы, связанные с проектированием отдельных реляционных предметных БД. Основная цель проектирования БД – это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте. Так называемый, «чистый» проект БД («Каждый факт в одном месте») можно создать, используя методологию нормализации отношений. И хотя нормализация должна использоваться на завершающей проверочной стадии проектирования БД, мы начнем обсуждение вопросов проектирования с рассмотрения причин, которые заставили Кодда создать основы теории нормализации.

Этапы проектирования базы данных.

Целью разработки любой базы данных является хранение и использование информации о какой-либо предметной области. Для реализации этой цели имеются следующие инструменты:

1) Реляционная модель данных - удобный способ представления данных предметной области.

2) Язык SQL - универсальный способ манипулирования такими данными.

Однако очевидно, что для одной и той же предметной области реляционные отношения можно спроектировать множеством различных способов. Например, можно спроектировать несколько отношений с большим количеством атрибутов, или наоборот, разнести все атрибуты по большому числу мелких отношений. Как определить, по каким признакам нужно помещать атрибуты в те или иные отношения?

В данной главе рассматриваются способы «хорошего» или «правильного» проектирования реляционных отношений. Сначала мы обсудим, что значит «хорошие» или «правильные» модели данных. Потом будут введены понятия первой, второй и третьей нормальных форм отношений (1НФ, 2НФ, 3НФ), и показано, что «хорошими» являются отношения в третьей нормальной форме.

При разработке базы данных обычно выделяется несколько уровней моделирования, при помощи которых происходит переход от предметной области к конкретной реализации базы данных средствами конкретной СУБД. Можно выделить следующие уровни:

1) Сама предметная область.

2) Модель предметной области.

3) Логическая модель данных.

4) Физическая модель данных.

5) Собственно база данных и приложения.

Предметная область - это часть реального мира, данные о которой мы хотим отразить в базе данных. Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т.д. Предметная область бесконечна и содержит как существенно важные понятия и данные, так и малозначащие или вообще не значащие данные. Так, если в качестве предметной области выбрать учет товаров на складе, то понятия «накладная» и «счет-фактура» являются существенно важными понятиями, а то, что сотрудница, принимающая накладные, имеет двоих детей - это для учета товаров неважно. Однако с точки зрения отдела кадров данные о наличии детей являются существенно важными. Таким образом, важность данных зависит от выбора предметной области.

Модель предметной области. Модель предметной области - это наши знания о предметной области. Знания могут быть как в виде неформальных знаний в мозгу эксперта, так и выражены формально при помощи каких-либо

средств. В качестве таких средств могут выступать текстовые описания предметной области, наборы должностных инструкций, правила ведения дел в компании и т.п. Опыт показывает, что текстовый способ представления модели предметной области крайне неэффективен. Гораздо более информативными и полезными при разработке баз данных являются описания предметной области, выполненные при помощи специализированных графических нотаций. Имеется большое количество методик описания предметной области. Из наиболее известных можно назвать методику структурного анализа SADT и основанную на нем IDEF0, диаграммы потоков данных Гейна-Сарсона, методику объектно-ориентированного анализа UML, и др. Модель предметной области описывает скорее процессы, происходящие в предметной области и данные, используемые этими процессами. От того, насколько правильно смоделирована предметная область, зависит успех дальнейшей разработки приложений.

Логическая модель данных. На следующем, более низком уровне находится логическая модель данных предметной области. Логическая модель описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью. Примеры понятий – «сотрудник», «отдел», «проект», «зарплата». Примеры взаимосвязей между понятиями – «сотрудник числится ровно в одном отделе», «сотрудник может выполнять несколько проектов», «над одним проектом может работать несколько сотрудников». Примеры ограничений – «возраст сотрудника не менее 18 и не более 60 лет».

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но *без привязки к конкретной СУБД*. Более того, логическая модель данных необязательно должна быть выражена средствами именно *реляционной* модели данных. Основным средством разработки логической модели данных в настоящий момент являются различные варианты *ER-диаграмм* (*Entity-Relationship, диаграммы сущность-связь*). Одну и ту же ER-модель можно преобразовать как в реляционную модель данных, так и в модель данных для иерархических и сетевых СУБД, или в постреляционную модель данных. Однако, т.к. мы рассматриваем именно реляционные СУБД, то можно считать, что логическая модель данных для нас формулируется в терминах реляционной модели данных. Решения, принятые на предыдущем уровне, при разработке модели предметной области, определяют некоторые границы, в пределах которых можно развивать логическую модель данных, в пределах же этих границ можно принимать различные решения. Например, модель предметной области складского учета содержит понятия «склад», «накладная», «товар». При разработке соответствующей реляционной модели эти термины обязательно должны быть использованы, но различных способов реализации тут много - можно создать одно отношение, в котором будут присутствовать в качестве атрибутов «склад», «накладная», «товар», а можно создать три отдельных отношения, по одному на каждое понятие. При

разработке логической модели данных возникают вопросы: хорошо ли спроектированы отношения? Правильно ли они отражают модель предметной области, а следовательно и саму предметную область?

Собственно база данных и приложения. И, наконец, как результат предыдущих этапов появляется собственно сама база данных. База данных реализована на конкретной программно-аппаратной основе, и выбор этой основы позволяет существенно повысить скорость работы с базой данных. Например, можно выбирать различные типы компьютеров, менять количество процессоров, объем оперативной памяти, дисковые подсистемы и т.п. Очень большое значение имеет также настройка СУБД в пределах выбранной программно-аппаратной платформы. Но опять решения, принятые на предыдущем уровне - уровне физического проектирования, определяют границы, в пределах которых можно принимать решения по выбору программно-аппаратной платформы и настройки СУБД. Таким образом ясно, что решения, принятые на каждом этапе моделирования и разработки базы данных, будут сказываться на дальнейших этапах. Поэтому особую роль играет принятие правильных решений *на ранних этапах моделирования*.

Адекватность базы данных предметной области

База данных должна адекватно отражать предметную область. Это означает, что должны выполняться следующие условия:

1) Состояние базы данных в каждый момент времени должно соответствовать состоянию предметной области.

2) Изменение состояния предметной области должно приводить к соответствующему изменению состояния базы данных

3) Ограничения предметной области, отраженные в модели предметной области, должны некоторым образом отражаться и учитываться базе данных.

Легкость разработки и сопровождения базы данных

Практически любая база данных, за исключением совершенно элементарных, содержит некоторое количество программного кода в виде триггеров и хранимых процедур.

Хранимые процедуры - это процедуры и функции, хранящиеся непосредственно в базе данных в откомпилированном виде, и которые могут запускаться пользователями или приложениями, работающими с базой данных. Хранимые процедуры обычно пишутся либо на специальном процедурном расширении языка SQL (например, PL/SQL для ORACLE или Transact-SQL для MS SQL Server), или на некотором универсальном языке программирования, например, C++, с включением в код операторов SQL в соответствии со специальными правилами такого включения. Основное назначение хранимых процедур - реализация бизнес-процессов предметной области.

Триггеры - это хранимые процедуры, связанные с некоторыми событиями, происходящими во время работы базы данных. В качестве таких событий выступают операции вставки, обновления и удаления строк таблиц. Если в базе данных определен некоторый триггер, то он запускается

автоматически всегда при возникновении события, с которым этот триггер связан. Очень важным является то, что пользователь не может обойти триггер. Триггер срабатывает независимо от того, кто из пользователей и каким способом инициировал событие, вызвавшее запуск триггера. Таким образом, основное назначение триггеров - автоматическая поддержка целостности базы данных. Триггеры могут быть как достаточно простыми, например, поддерживающими ссылочную целостность, так и довольно сложными, реализующими какие-либо сложные ограничения предметной области или сложные действия, которые должны произойти при наступлении некоторых событий. Например, с операцией вставки нового товара в накладную может быть связан триггер, который выполняет следующие действия - проверяет, есть ли необходимое количество товара, при наличии товара добавляет его в накладную и уменьшает данные о наличии товара на складе; при отсутствии товара формирует заказ на поставку недостающего товара и тут же посылает заказ по электронной почте поставщику.

Скорость операций обновления данных (вставка, обновление, удаление)

На уровне логического моделирования мы определяем реляционные отношения и атрибуты этих отношений. На этом уровне мы не можем определять какие-либо физические структуры хранения (индексы, хеширование и т.п.). Единственное, чем мы можем управлять - это распределением атрибутов по различным отношениям. Можно описать мало отношений с большим количеством атрибутов, или много отношений, каждое из которых содержит мало атрибутов. Таким образом, необходимо попытаться ответить на вопрос - влияет ли количество отношений и количество атрибутов в отношениях на скорость выполнения операций обновления данных. Такой вопрос, конечно, не является достаточно корректным, т.к. скорость выполнения операций с базой данных сильно зависит от физической реализации базы данных. Тем не менее, попытаемся *качественно* оценить это влияние при *одинаковых подходах к физическому моделированию*.

Основными операциями, изменяющими состояние базы данных, являются операции вставки, обновления и удаления записей. В базах данных, требующих постоянных изменений (складской учет, системы продаж билетов и т.п.) производительность определяется скоростью выполнения большого количества небольших операций вставки, обновления и удаления. Рассмотрим операцию вставки записи в таблицу. Вставка записи производится в одну из свободных страниц памяти, выделенной для данной таблицы. СУБД постоянно хранит информацию о наличии и расположении свободных страниц. Если для таблицы не созданы индексы, то операция вставки выполняется фактически с одинаковой скоростью независимо от размера таблицы и от количества атрибутов в таблице. Если в таблице имеются индексы, то при выполнении операции вставки записи индексы должны быть перестроены. Таким образом, скорость выполнения операции вставки *уменьшается при увеличении количества индексов* у таблицы и *мало зависит от числа строк* в таблице.

Рассмотрим операции обновления и удаления записей из таблицы. Прежде, чем обновить или удалить запись, ее необходимо найти. Если таблица не индексирована, то единственным способом поиска является последовательное сканирование таблицы в поиске нужной записи. В этом случае, скорость операций обновления и удаления существенно увеличивается с увеличением количества записей в таблице и не зависит от количества атрибутов. Но, на самом деле, неиндексированные таблицы практически никогда не используются. Для каждой таблицы обычно объявляется один или несколько индексов, соответствующий потенциальным ключам. При помощи этих индексов поиск записи производится очень быстро и практически не зависит от количества строк и атрибутов в таблице (хотя, конечно, некоторая зависимость имеется). Если для таблицы объявлено несколько индексов, то при выполнении операций обновления и удаления эти индексы должны быть перестроены, на что тратится дополнительное время. Таким образом, скорость выполнения операций обновления и удаления также *уменьшается при увеличении количества индексов* у таблицы и *мало зависит от числа строк* в таблице. Можно предположить, что, чем больше атрибутов имеет таблица, тем больше для нее будет объявлено индексов. Эта зависимость, конечно, не прямая, но *при одинаковых подходах* к физическому моделированию обычно так и происходит. Таким образом, можно принять допущение, что *чем больше атрибутов имеют отношения*, разработанные в ходе логического моделирования, *тем медленнее будут выполняться операции обновления данных*, за счет затраты времени на перестройку большего количества индексов.

Скорость операций выборки данных.

Одно из назначений базы данных - предоставление информации пользователям. Информация извлекается из реляционной базы данных при помощи оператора SQL - SELECT. Одной из наиболее дорогостоящих операций при выполнении оператора SELECT является операция соединение таблиц. Таким образом, чем больше взаимосвязанных отношений было создано в ходе логического моделирования, тем больше вероятность того, что при выполнении запросов эти отношения будут соединяться, и, следовательно, тем медленнее будут выполняться запросы. Таким образом, увеличение количества отношений приводит к замедлению выполнения операций выборки данных, особенно, если запросы заранее неизвестны.

Контрольные вопросы:

1) Что такое предметные базы данных?
2) В чем заключается цель базы данных?
3) Перечислите средства СУБД?
4) Что является инструментом БД ?
5) Что такое предметная область и модель предметной области?

Лекция №6. Модель «Сущность-связь». Логические и физические модели

Как отмечалось, инфологическая 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». Инфологическая модель должна быть отображена в компьютеро-ориентированную даталогическую модель, «понятную» СУБД.

В процессе развития теории и практического использования баз данных, а также средств вычислительной техники создавались СУБД, поддерживающие различные даталогические модели. Сначала стали использовать иерархические даталогические модели. Простота организации, наличие заранее заданных связей между сущностями, сходство с физическими моделями данных позволяли добиваться приемлемой производительности иерархических СУБД на медленных ЭВМ с весьма ограниченными объемами памяти. Но, если данные не имели древовидной структуры, то возникала масса сложностей при построении иерархической модели и желании добиться нужной производительности. Сетевые модели также создавались для мало ресурсных ЭВМ. Это достаточно сложные структуры, состоящие из «наборов» – поименованных двухуровневых деревьев. «Наборы» соединяются с помощью «записей-связок», образуя цепочки и т.д. При разработке сетевых моделей было выдумано множество «маленьких хитростей», позволяющих увеличить производительность СУБД, но существенно усложнивших последние. Прикладной программист должен знать массу терминов, изучить несколько внутренних языков СУБД, детально представлять логическую структуру базы данных для осуществления навигации среди различных экземпляров, наборов, записей и т.п. Один из разработчиков операционной системы UNIX сказал «Сетевая база – это самый верный способ потерять данные». Сложность практического использования иерархических и сетевых СУБД заставляла искать иные способы представления данных. В конце 60-х годов появились СУБД на основе инвертированных файлов, отличающиеся простотой организации и наличием весьма удобных языков манипулирования данными. Однако такие СУБД обладают рядом ограничений на количество файлов для хранения данных, количество связей между ними, длину записи и количество ее полей. Сегодня наиболее распространены реляционные модели.

Физическая организация данных оказывает основное влияние на эксплуатационные характеристики БД. Разработчики СУБД пытаются создать наиболее производительные физические модели данных, предлагая пользователям тот или иной инструментарий для поднастройки модели под конкретную БД. Разнообразие способов корректировки физических моделей современных промышленных СУБД не позволяет рассмотреть их в этом разделе.

Моделирование локальных представлений, агрегация и обобщение элементов моделей.

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным языком (последний не может быть использован в чистом виде из-за сложности компьютерной обработки текстов и неоднозначности любого естественного языка). Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

Сущность – любой различимый объект (объект, который мы можем отличить от другого), информацию о котором необходимо хранить в базе данных. Сущностями могут быть люди, места, самолеты, рейсы, вкус, цвет и т.д. Необходимо различать такие понятия, как *тип сущности* и *экземпляр сущности*. Понятие тип сущности относится к набору однородных личностей, предметов, событий или идей, выступающих как целое. Экземпляр сущности относится к конкретной вещи в наборе. Например, типом сущности может быть ГОРОД, а экземпляром – Москва, Киев и т.д.

Атрибут – поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей (например, ЦВЕТ может быть определен для многих сущностей: СОБАКА, АВТОМОБИЛЬ, ДЫМ и т.д.). Атрибуты используются для определения того, какая информация должна быть собрана о сущности. Примерами атрибутов для сущности АВТОМОБИЛЬ являются ТИП, МАРКА, НОМЕРНОЙ ЗНАК, ЦВЕТ и т.д. Здесь также существует различие между типом и экземпляром. Тип атрибута ЦВЕТ имеет много экземпляров или значений:

Красный, Синий, Банановый, Белая ночь и т.д.,однако, каждому экземпляру сущности присваивается только одно значение атрибута. Абсолютное различие между типами сущностей и атрибутами отсутствует. Атрибут является таковым только в связи с типом сущности. В другом контексте атрибут может выступать как самостоятельная сущность. Например, для автомобильного завода цвет – это только атрибут продукта производства, а для лакокрасочной фабрики цвет – тип сущности.

Ключ – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Для сущности «Расписание» ключом является атрибут «Номер_рейса» или набор: «Пункт_отправления», «Время_вылета» и «Пункт_назначения», (при условии, что из пункта в пункт вылетает в каждый момент времени один самолет).

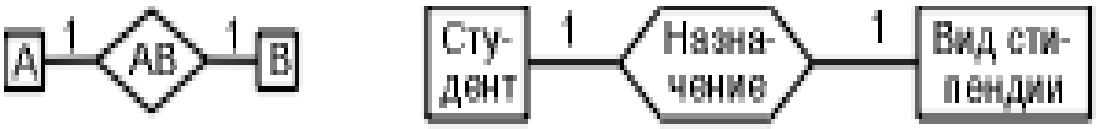
Связь – ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных – это обеспечение возможности отыскания одних сущностей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

Характеристика связей и язык моделирования.

При построении инфологических моделей можно использовать язык *ER-диаграмм* (от англ. Entity-Relationship, т.е. сущность-связь). В них сущности изображаются помеченными прямоугольниками, ассоциации – помеченными ромбами или шестиугольниками, атрибуты – помеченными овалами, а связи между ними – ненаправленными ребрами, над которыми может проставляться степень связи (1 или буква, заменяющая слово «много») и необходимое пояснение.

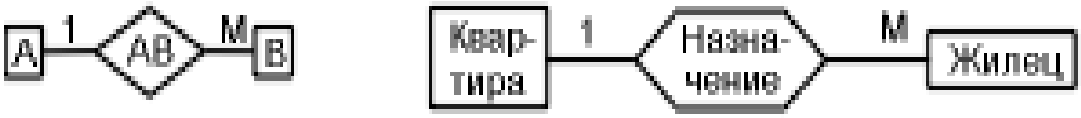
Между двумя сущностям, например, А и В возможны четыре вида связей.

Первый тип – связь ОДИН-К-ОДНОМУ (1:1): в каждый момент времени каждому представителю (экземпляру) сущности А соответствует 1 или 0 представителей сущности В:



Студент может не «заработать» стипендию, получить обычную или одну из повышенных стипендий.

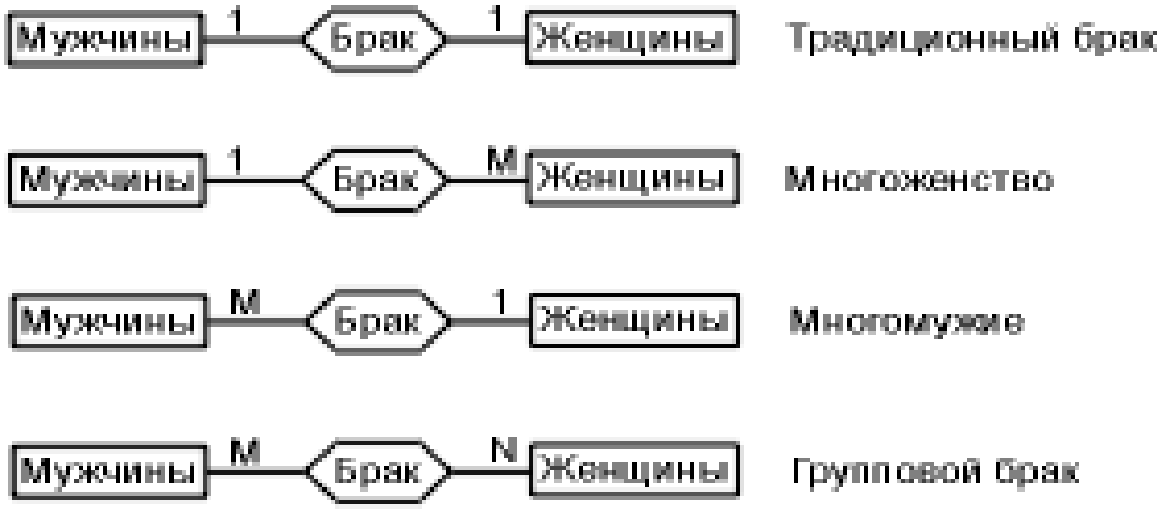
Второй тип – связь ОДИН-КО-МНОГИМ (1:M): одному представителю сущности А соответствуют 0, 1 или несколько представителей сущности В.



Квартира может пустовать, в ней может жить один или несколько жильцов.

Так как между двумя сущностями возможны связи в обоих направлениях, то существует еще два типа связи МНОГИЕ-К-ОДНОМУ (M:1) и МНОГИЕ-КО-МНОГИМ (M:N).

Пример 2.1. Если связь между сущностями МУЖЧИНЫ и ЖЕНЩИНЫ называется БРАК, то существует четыре возможных представления такой связи.

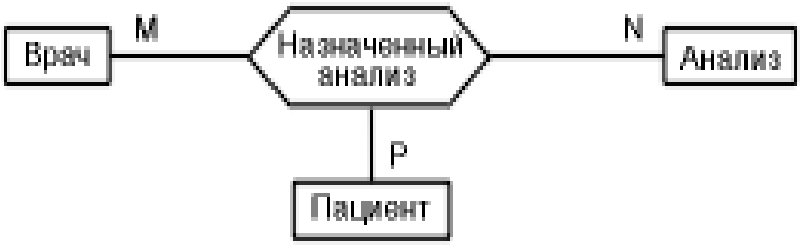


Характер связей между сущностями не ограничивается перечисленными. Существуют и более сложные связи:

- множество связей между одними и теми же сущностями (пациент, имея одного лечащего врача, может иметь также несколько врачей-консультантов; врач может быть лечащим врачом нескольких пациентов и может одновременно консультировать несколько других пациентов);

- тренарные связи (врач может назначить несколько пациентов на несколько анализов, анализ может быть назначен несколькими врачами нескольким пациентам и пациент может быть назначен на несколько анализов несколькими врачами);

- связи более высоких порядков, семантика (смысл) которых иногда очень сложна.



Контрольные вопросы:

1) Объясните сущность модели «Сущность и связь».

2) Что такое «сущность»?

3) Что определяет «атрибут»?
4) Что такое «Ключ» ?
5) Перечислите и опишите классификацию сущностей.

Лекция №7. Модели баз данных. Сетевые и иерархические модели данных. Реляционная модель данных. Проектирование реляционных БД

На разработку этого стандарта большое влияние оказал американский ученый Ч. Бахман. Основные принципы сетевой модели данных были разработны в середине 60-х годов, эталонный вариант сетевой модели данных описан в отчетах рабочей группы по языкам баз данных (COnference on DAta SYstem Languages) CODASYL (1971 г.). 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 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. Согласно этой модели каждое групповое отношение именуется и проводится различие между его типом и экземпляром. Тип группового отношения задается его именем и определяет свойства общие для всех экземпляров данного типа. Экземпляр группового отношения представляется записью-владельцем и множеством (возможно пустым) подчиненных записей. При этом имеется следующее ограничение: экземпляр записи не может быть членом двух экземпляров групповых отношений одного.

Иерархическая структура преобразовывается в сетевую следующим образом:

- деревья (a) и (b), показанные на рисунке 7, заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в два групповых отношения;

- для отображения типа M:N вводится запись СОТРУДНИК_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК (рисунок 7), (отметим, что в этой записи может храниться и полезная информация, например, доля данного сотрудника в общем вознаграждении по данному контракту).

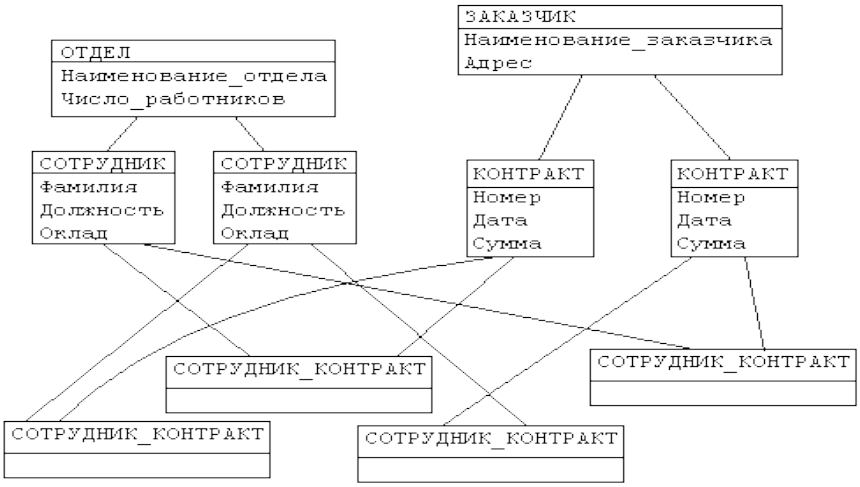


Рисунок 7 - Сетевая модель

Каждый экземпляр группового отношения характеризуется следующими признаками:

- способ упорядочения подчиненных записей:
- произвольный;
- хронологический /очередь/;
- обратный хронологический /стек/;
- сортированный.

Если запись объявлена подчиненной в нескольких групповых отношениях, то в каждом из них может быть назначен свой способ упорядочивания.

- режим включения подчиненных записей:
- автоматический - невозможно занести в БД запись без того, чтобы она была сразу же закреплена за неким владельцем;
- ручной - позволяет запомнить в БД подчиненную запись и не включать ее немедленно в экземпляр группового отношения. Эта операция позже инициируется пользователем).
- режим исключения Принято выделять три класса членства подчиненных записей в групповых отношениях:

1) Фиксированное. Подчиненная запись жестко связана с записью владельцем, и ее можно исключить из группового отношения, только удалив. При удалении записи-владельца все подчиненные записи автоматически тоже удаляются. В рассмотренном выше примере фиксированное членство предполагает групповое отношение «ЗАКЛЮЧАЕТ» между записями «КОНТРАКТ» и «ЗАКАЗЧИК», поскольку контракт не может существовать без заказчика.

2) Обязательное. Допускается переключение подчиненной записи на другого владельца, но невозможно ее существование без владельца. Для

удаления записи-владельца необходимо, чтобы она не имела подчиненных записей с обязательным членством. Таким отношением связаны записи «СОТРУДНИК» и «ОТДЕЛ». Если отдел расформировывается, все его сорудники должны быть либо переведены в другие отделы, либо уволены.

3) Необязательное. Можно исключить запись из группового отношения, но сохранить ее в базе данных не прикрепляя к другому владельцу. При удалении записи-владельца ее подчиненные записи - необязательные члены сохраняются в базе, не участвуя более в групповом отношении такого типа. Примером такого группового отношения может служить «ВЫПОЛНЯЕТ» между «СОТРУДНИКИ» и «КОНТРАКТ», поскольку в организации могут существовать работники, чья деятельность не связана с выполненинем каких-либо договорных обязательств перед заказчиками.

Операции над данными:

- «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.

Ограничения целостности.

Как и в иерархической модели, обеспечивается только поддержание целостности по ссылкам (владелец отношения - член отношения).

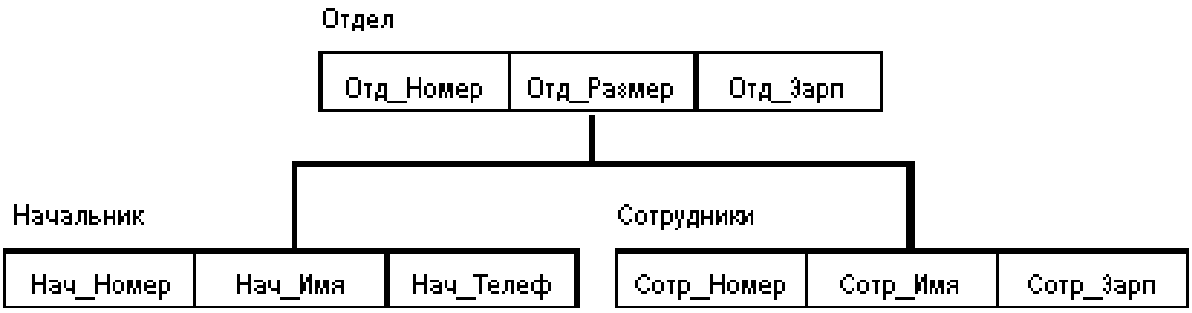
Иерархическая модель.

Типичным представителем (наиболее известным и распространенным) является Information Management System (IMS) фирмы IBM. Первая версия появилась в 1968 г. До сих пор поддерживается много баз данных, что создает существенные проблемы с переходом как на новую технологию БД, так и на новую технику.

Иерархические структуры данных.

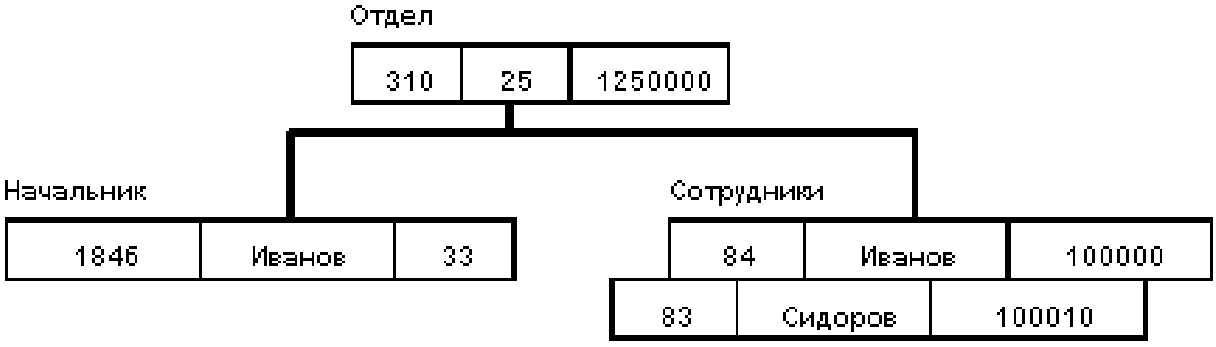
Иерархическая БД состоит из упорядоченного набора деревьев; более точно, из упорядоченного набора нескольких экземпляров одного типа дерева. Тип дерева состоит из одного «корневого» типа записи и упорядоченного набора из нуля или более типов поддеревьев (каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор типов записи.

Пример типа дерева (схемы иерархической БД):



Здесь Отдел является предком для Начальник и Сотрудники, а Начальник и Сотрудники - потомки Отдел. Между типами записи поддерживаются связи.

База данных с такой схемой могла бы выглядеть следующим образом (мы показываем один экземпляр дерева):



Все экземпляры данного типа потомка с общим экземпляром типа предка называются близнецами. Для БД определен полный порядок обхода - сверху-вниз, слева-направо.

В IMS использовалась оригинальная и нестандартная терминология: «сегмент» вместо «запись», а под «записью БД» понималось все дерево сегментов.

Манипулирование данными.

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

1) Найти указанное дерево БД (например, отдел 310).

2) Перейти от одного дерева к другому.

3) Перейти от одной записи к другой внутри дерева (например, от отдела - к первому сотруднику).

4) Перейти от одной записи к другой в порядке обхода иерархии.

5) Вставить новую запись в указанную позицию.

6) Удалить текущую запись.

Ограничения целостности.

Автоматически поддерживается целостность ссылок между предками и потомками. *Основное правило: никакой потомок не может существовать без своего родителя.* Заметим, что аналогичное поддержание целостности по ссылкам между записями, не входящими в одну иерархию, не поддерживается (примером такой «внешней» ссылки может быть содержимое поля «Каф_Номер» в экземпляре типа записи «Куратор»).

В иерархических системах поддерживалась некоторая форма представлений БД на основе ограничения иерархии. Примером представления приведенной выше БД может быть иерархия

Сетевые структуры данных.

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах «запись-потомок» должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков.

Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно, из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

1) Каждый экземпляр типа P является предком только в одном экземпляре L.

2) Каждый экземпляр C является потомком не более, чем в одном экземпляре L.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

1) Тип записи потомка в одном типе связи L1 может быть типом записи предка в другом типе связи L2 (как в иерархии).

2) Данный тип записи P может быть типом записи предка в любом числе типов связи.

3) Данный тип записи P может быть типом записи потомка в любом числе типов связи.

4) Может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если L1 и L2 - два типа связи с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться.

5) Типы записи X и Y могут быть предком и потомком в одной связи и потомком и предком - в другой.

6) Предок и потомок могут быть одного типа записи.

Контрольные вопросы:

1) Опишите иерархическую структуру данных.

2) Основное правило ограничения целостности.

3) Опишите сетевую структуру данных.

4) Опишите иерархическую модель данных.

5) Какие термины используются в иерархической модели представления данных?

6) Как описывается объект в терминах иерархической модели?

7) Каковы особенности выполнения операций включения и удаления в иерархической модели?

8) Какие термины используются в сетевой модели представления данных?

Лекция №8. Нормализация отношений

Нормализация отношений представляет собой процесс построения оптимальной структуры отношений и связей в реляционной базе данных. Теория нормализации основана на том, что определенное множество отношений обладает лучшими свойствами при добавлении, корректировке и удалении данных, чем другие множества отношений, с помощью которых могут быть представлены те же данные. *E. Codd* первоначально определил три уровня нормализации, которые назвал 1-ой, 2-ой и 3-ей нормальными формами (1НФ, 2НФ, 3НФ). Наличие различных зависимостей в отношении приводит к сложности выполнения операций по обработке данных.

Определим эти нормальные формы. Нормализованным называется отношение, каждый домен которого содержит только атомарные (неделимые) значения, и поэтому каждое значение в отношении в свою очередь является

атомарным. Отношение *R* находится в 1-ой нормальной форме, если ни один из атрибутов, входящих в него, не является множеством, т.е. между атрибутами нет никакой зависимости.

Введем понятие функциональной зависимости. Атрибут *B* отношения *R* функционально зависит от атрибута *A*, принадлежащего *R*, тогда и только тогда, когда каждое значение *A* в отношении *R* в каждый момент времени связано точно с одним значением *B*: $R.A \rightarrow R.B$. Если между атрибутами отношения выявлена функциональная зависимость, то избавиться от нее можно путем разбиения исходного отношения на два и более отношений, которые будут находиться во 2-ой нормальной форме. Рассмотрим следующий пример. Пусть имеется отношение *R*, включающее в свой состав следующие атрибуты: A_1 - фамилия студента, A_2 - выпускающая кафедра, A_3 - местоположение кафедры. В таблице 2 представлен экземпляр этого отношения.

Таблица 2 - Отношение *R*

A_1	A_2	A_3
Ахметов	Техническая экспертиза и оценка	1035
Каримов	Менеджмент и маркетинг	808
Петров	Техническая экспертиза и оценка	1035
Оспанов	Менеджмент и маркетинг	808
Ли	Техническая экспертиза и оценка	1035

Как видно из таблицы 2, между атрибутами A_2 и A_3 отношения *R* существует функциональная зависимость. Избавиться от нее можно, разбив исходное отношение на следующие два отношения: $R_1(A_1, A_2)$ и $R_2(A_2, A_3)$. Экземпляры этих отношений приведены в таблицах 3 и 4.

Таблица 3 - Отношение R_1

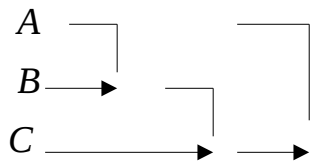
A_1	A_2
Ахметов	Техническая экспертиза и оценка
Каримов	Менеджмент и маркетинг
Петров	Техническая экспертиза и оценка
Оспанов	Менеджмент и маркетинг
Ли	Техническая экспертиза и оценка

Таблица 4 - Отношение R_2

A_2	A_3
Техническая экспертиза и оценка	1035
Менеджмент и маркетинг	808

Из таблицы 4 видно, что если изменится местоположение кафедры, то провести корректировку проще в отношении R_2 , чем в исходном отношении *R*.

Введем понятие транзитивной зависимости. Пусть *A, B, C* – атрибуты отношения *R*. Если *C* зависит от *B*, а *B* зависит от *A* и при этом обратное соответствие неоднозначно (т.е. *A* не зависит от *B* или *B* не зависит от *C*), то говорят, что *C* транзитивно зависит от *A*:



Если между атрибутами отношения выявлена транзитивная зависимость, то избавиться от нее можно путем разбиения исходного отношения на два и более отношений, которые будут находиться во 3-ей нормальной форме. Рассмотрим следующий пример. Пусть имеется отношение *R*, включающее в свой состав следующие атрибуты: A_1 - номер зачетки (уникальный ключ), A_2 - фамилия студента, A_3 - дата рождения, A_4 - номер группы, A_5 - фамилия старосты, A_6 - адрес старосты. В таблице 5 представлен экземпляр этого отношения.

Таблица 5 - Отношение *R*

A_1	A_2	A_3	A_4	A_5	A_6
112356	Ахметов	11.03.85	ЭК-98	Аманов	Абая 22-15
213478	Каримов	25.01.84	ЭП-99	Иванов	Сатпаева 3-20
146732	Петров	03.05.84	ЭК-98	Аманов	Абая 22-15
123155	Оспанов	09.06.85	ЭП-99	Иванов	Сатпаева 3-20
120176	Ли	07.11.84	ЭК-98	Аманов	Абая 22-15

Как видно из таблицы 5, атрибуты A_2 , A_3 , A_4 отношения *R* зависят от ключевого атрибута A_1 **,** а атрибуты A_5 , A_6 зависят от A_4 , т.е. между атрибутами существует транзитивная зависимость. Избавиться от нее можно, разбив исходное отношение на следующие два отношения: $R_1(A_1, A_2, A_3, A_4)$ и $R_2(A_4, A_5, A_6)$. Экземпляры этих отношений приведены в таблицах 6 и 7.

Таблица 6 - Отношение R_1

A_1	A_2	A_3	A_4
112356	Ахметов	11.03.85	ЭК-98
213478	Каримов	25.01.84	ЭП-99
146732	Петров	03.05.84	ЭК-98

123155	Оспанов	09.06.85	ЭП-99
120176	Ли	07.11.84	ЭК-98

Таблица 7 - Отношение R_2

A_4	A_5	A_6
ЭК-98	Аманов	Абая 22-15
ЭП-99	Иванов	Сатпаева 3-20

В настоящее время в теории нормализации введены 4-я, 5-я и 6-я нормальные формы (4НФ, 5НФ, 6НФ). Процесс нормализации включает следующие шаги:

- представление объекта в виде отношения в 1-й нормальной форме;
- устранение функциональной зависимости путем перехода из 1НФ в 2НФ;
- устранение транзитивной зависимости путем перехода из 2НФ в 3НФ;
- устранение других зависимостей путем перехода в 4НФ, 5НФ и 6НФ.

Контрольные вопросы:

1) Какие понятия используются в реляционной модели?

2) Как представляется объект и его свойства в реляционной модели данных?

3) Что такое функциональная зависимость?

4) Что такое транзитивная зависимость?

5) Какие шаги включает алгоритм нормализации отношений?

Лекция №9. Реляционная алгебра и реляционное исчисление

Систему операций, используемую для манипулирования отношениями, называют реляционной алгеброй (РА). Операция в РА имеет одно или два отношения в качестве операндов и образует новое отношение по определенному правилу. Реляционная алгебра включает 6 операций: перестановка, проекция, соединение, ограничение, композиция, деление. Рассмотрим эти операции.

1) Перестановка позволяет изменить порядок следования атрибутов в отношении. Пусть *R* – отношение, включающее *n* атрибутов. $L=\{i_1,i_2,\ldots,i_n\}$ - новый порядок следования атрибутов, i_j , $j=\overline{1,n}$ - номер атрибута из *R*.

$S=R[L]=\{r[L]!r\in R\}$ - новое отношение.

2) Проекция позволяет выбрать из отношения атрибуты, указанные в списке *L*. Пусть *R* – отношение, включающее *n* атрибутов. $L=\{i_1,i_2,\ldots,i_k\}$ - новый порядок следования атрибутов, i_j , $j=\overline{1,k}$, $k < n$ - номер атрибута из *R*.

$S=R[L]=\{r[L]!r\in R\}$ - новое отношение.

3) Соединение позволяет получить новое отношение из 2-х исходных отношений (*R* и *S*), имеющих одинаковые атрибуты (*A* и *B)*, по которым и происходит соединение строк отношений.

$Q=R[A\Theta B]S=\{(r,s)!r\in R\wedge s\in S\wedge(r[A]\Theta s[B])\}$ - новое отношение,

$\Theta=$ - знаки сравнения.

4) Ограничение позволяет выбрать из исходного отношения *R* только те строки, для которых выполняются условия, указанные в отношении *S*.

$Q=R[A\Theta B]S=\{r)!r\in R\wedge(r[A]\Theta s[B])\}$ - новое отношение.

5) Композиция аналогична «соединению», но в новое отношение не входят атрибуты, по которым проходило соединение.

$Q=R[A\Theta B]S=\{(r^1,s^1)!r^1\in R\wedge s^1\in S\wedge(r[A]\Theta s[B],r[A]\notin r^1,s[B]\notin s^1)\}$ - новое отношение.

6) Деление выполняется над двумя исходными отношениями (*R* и *S*), новое отношение формируется по следующей формуле:

$Q=R[A\div B]S=\{r[\overline{A}]!r\in R\wedge y!\{(r[\overline{A}],y\in R\}\Theta s[B]\}$.

Рассмотрим примеры выполнения некоторых операций. Пусть имеется отношение *R*, имеющее атрибуты A_1 , A_2 , A_3 и представленное в таблице 9. Для операции «проекция» задан новый порядок *L={3,1}*. Новое отношение *Q*, полученное как результат «проекции», представлено в таблице 9.

Таблица 8 - Отношение *R*

A_1	A_2	A_3
2	11	103
3	15	112
7	16	120

Таблица 9 - Отношение *Q*

A_3	A_1
103	2
112	3
120	7

Рассмотрим выполнение операции «соединение». Исходными являются отношение *R* (таблица 8) и отношение *S* (таблица 10). Сравнение будет идти по знаку $\Theta=\{\geq\}$ (больше или равно). Новое отношение представлено в таблице 11.

Таблица 10 - Отношение *S*

A_3	A_5
105	552

120	563

Таблица 11 - Отношение *Q*

A_1	A_2	A_3	A_3	A_5
3	15	112	105	552
7	16	120	105	552
7	16	120	120	563

Реляционное исчисление (РИ) позволяет описать отношение и операции над ним в виде аналитического выражения или формулы путем использования ограниченного исчисления предикатов, кванторов и переменных выборки. Пользователь просто указывает, что он хочет получить из БД. Для построения выражений используются следующие символы:

1) *x.y* – множество значений атрибута *y* из отношения *x*.

2) $A(x_1.y_1,x_2.y_2,\ldots)$ - отношение *A*, заданное с указанными атрибутами.

3) : - «Такой, что», выражение слева от двоеточия означает то, что должно быть найдено, выражение справа означает условие поиска данных.

4) $\exists$ - квантор существования («существует»).

5) $\forall$ - квантор всеобщности («для любого»).

6) $\wedge, \vee, \neg$ - логические операции «И», «ИЛИ», «НЕ».

7) $=, \neq, >, \geq, <, \leq$ - знаки сравнения.

Приведем примеры использования реляционного исчисления. Пусть имеется отношение *R*, включающее в свой состав следующие атрибуты: A_1 - номер служащего (уникальный ключ), A_2 - фамилия служащего, A_3 - номер отдела, в котором он работает, A_4 - зарплата служащего.

Пример 1 – создать новое отношение, содержащее фамилии всех служащих, работающих в отделе с номером 3: $Q\,(R.\,A_2\,) : R.\,A_3 = 3$.

Пример 2 – создать новое отношение, содержащее номера и фамилии всех служащих, работающих в отделе номер 3 с зарплатой больше 8000:

$$Z\,(R.\,A_1\,, R.\,A_2\,) : R.\,A_3 = 3 \wedge R.\,A_4 > 8000 \quad .$$

Проведем сравнение реляционной алгебры и реляционного исчисления. Множество функций отношений, выразимых в РА, являются в точности тем же самым, что и множество функций, выразимых формулами РИ. Языки, основанные на РИ, представляют собой языки более высокого уровня по сравнению с алгебраическими языками, поскольку алгебра специфицирует порядок операций в то время, как РИ оставляет определение более эффективного порядка вычислений программному пакету. Выигрыш получают, если выражение РИ оптимизировано, что является весьма сложной проблемой. РИ требует более высокого уровня автоматизации, реализовать его

сложнее. Для пользователя, естественно, более удобно указать, что надо найти, чем как это искать.

Контрольные вопросы:

1) Какие операции есть в реляционной алгебре?

2) Как выполняется операция «проекция»?

3) Как выполняется операция «соединение»?

4) Какие символы используются в выражении реляционного исчисления?

5) Как записывается выражение в реляционном исчислении?

Лекция №10. Язык реляционных баз данных SQL

Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных - язык SQL (Structured Query Language). Язык для взаимодействия с БД SQL появился в середине 70-х годах.

Язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Специальные операторы языка SQL позволяют определять так называемые представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Поддержание представлений производится также на языковом уровне.

Авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает

полным набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволило создать компактный язык с небольшим (менее 30) набором предложений. SQL может использоваться как интерактивный (для выполнения запросов) и как встроенный (для построения прикладных программ). В нем существуют:

- предложения определения данных (определение баз данных, а также определение и уничтожение таблиц и индексов);
- запросы на выбор данных (предложение SELECT);
- предложения модификации данных (добавление, удаление и изменение данных);
- предложения управления данными (предоставление и отмена привилегий на доступ к данным, управление транзакциями и другие). Кроме того, он предоставляет возможность выполнять в этих предложениях:
- арифметические вычисления (включая разнообразные функциональные преобразования), обработку текстовых строк и выполнение операций сравнения значений арифметических выражений и текстов;
- упорядочение строк и (или) столбцов при выводе содержимого таблиц на печать или экран дисплея;
- создание представлений (виртуальных таблиц), позволяющих пользователям иметь свой взгляд на данные без увеличения их объема в базе данных;
- запоминание выводимого по запросу содержимого таблицы, нескольких таблиц или представления в другой таблице (реляционная операция присваивания).
- агрегатирование данных: группирование данных и применение к этим группам таких операций, как среднее, сумма, максимум, минимум, число элементов и т.п.

Типы данных SQL.

В SQL используются следующие основные типы данных, форматы которых могут несколько различаться для разных СУБД:

- INTEGER - целое число (обычно до 10 значащих цифр и знак);
- SMALLINT - "короткое целое" (обычно до 5 значащих цифр и знак);
- DECIMAL(p,q) - десятичное число, имеющее p цифр ($0 < p < 16$) и знак; с помощью q задается число цифр справа от десятичной точки ($q < p$, если $q = 0$, оно может быть опущено);
- FLOAT - вещественное число с 15 значащими цифрами и целочисленным порядком, определяемым типом СУБД;
- CHAR(n) - символьная строка фиксированной длины из n символов ($0 < n < 256$);

- VARCHAR(n) - символьная строка переменной длины, не превышающей n символов (n > 0 и разное в разных СУБД, но не меньше 4096);

- DATE - дата в формате, определяемом специальной командой (по умолчанию mm/dd/yy); поля даты могут содержать только реальные даты, начинающиеся за несколько тысячелетий до н.э. и ограниченные пятым-десятым тысячелетием н.э.;

- TIME - время в формате, определяемом специальной командой (по умолчанию hh.mm.ss);

- DATETIME - комбинация даты и времени;

- MONEY - деньги в формате, определяющем символ денежной единицы ($, руб, ...) и его расположение (суффикс или префикс), точность дробной части и условие для показа денежного значения.

Понятие «таблица», как правило, связывается с реальной или базовой таблицей, т.е. с таблицей, для каждой строки которой в действительности имеется некоторый двойник, хранящийся в физической памяти машины. Однако SQL использует и создает ряд виртуальных (как будто существующих) таблиц: представлений, курсоров и неименованных рабочих таблиц, в которых формируются результаты запросов на получение данных из базовых таблиц и, возможно, представлений. Это таблицы, которые не существуют в базе данных, но как бы существуют с точки зрения пользователя.

База данных во многих СУБД представляет собой контейнер, содержащий таблицы, представления (виртуальные таблицы), триггеры (для отслеживания ограничений целостности – непротиворечивости данных), хранимые процедуры и другие объекты.

Для создания базы данных используется команда:

CREATE DATABASE <имя_БД>

[необязательные_параметры].

Состав необязательных параметров зависит от используемой СУБД, при их отсутствии в команде используются значения по умолчанию.

Для определения структуры таблицы данных используются команды создания (CREATE TABLE) и изменения (ALTER TABLE) структуры таблицы. Таблица создается в активной БД. Если нет активной БД, то в СУБД, допускающих наличие свободных таблиц, создается свободная таблица (не входящая в состав БД).

Команда создания структуры таблицы данных имеет следующий формат:

CREATE TABLE <имя_таблицы> (<определение_поля1> [,<определение_поля2>...][определение_ограничений])

Команда создает новую таблицу данных с указанным именем. Для каждого поля задаются его имя и тип. Для типов полей, которые могут иметь разную ширину, дополнительно необходимо указать требуемую ширину. Для числовых полей с плавающей запятой указывается также точность (число десятичных разрядов). Если создаваемое поле будет являться ключевым

(индексом), то необходимо указать тип ключа (индекса). Кроме того, в определении поля можно указать признак (NOT NULL) обязательного заполнения поля при вводе данных в таблицу, значения полей по умолчанию и ограничения на ввод значений. Определение ограничений таблицы используется для задания ограничений на уровне таблицы, могут указываться также определения первичного и внешнего ключей.

В Visual FoxPro команда CREATE TABLE создает новую таблицу данных (DBF-файл) с указанным именем. Для каждого поля задаются его имя и тип (одной из букв: C - символьный; N - числовой; F - с плавающей запятой; D - дата; L - логический; M - примечание и т.д.).

Можно создать временную таблицу, с которой можно работать как с обычной таблицей данных, называемую курсором (CURSOR). Курсор располагается в оперативной памяти в свободной рабочей области и доступен сразу после создания до тех пор, пока он не будет закрыт. При закрытии курсора временная таблица удаляется из памяти. Команда создания курсора CREATE CURSOR аналогична команде CREATE TABLE.

Кроме того, с помощью команды CREATE можно создать индекс (CREATE INDEX), триггер (CREATE TRIGGER), представление (CREATE VIEW), значение по умолчанию (CREATE DEFAULT), ограничение на данные (CREATE RULE), хранимую процедуру (CREATE PROCEDURE), пользовательскую функцию (CREATE FUNCTION), схему данных (CREATE SCHEME).

При изменении структуры таблицы используется команда ALTER TABLE с ключевыми словами, которые позволяют добавить, удалить, переименовать поле, изменить параметры поля. Эта команда имеет формат:

ALTER TABLE <имя_таблицы>
[ADD <определение_добавляемого_поля>]
[DROP <имя_удаляемого_поля>]
[RENAME <старое_имя_поля> TO <новое_имя_поля>]
[ALTER <имя_изменяемого_поля>
<определение_изменяемых_параметров_поля>]

Кроме того, с помощью команды ALTER можно изменить триггер (ALTER TRIGGER), представление (ALTER VIEW), хранимую процедуру (ALTER PROCEDURE), пользовательскую функцию (ALTER FUNCTION).

С помощью команды DROP можно удалить любые объекты, созданные командой CREATE.

Можно удалить всю таблицу целиком командой:

DROP TABLE <имя_таблицы>

Для удаления базы данных со всеми таблицами используется команда:

DROP DATABASE <имя_базы_данных>

Дополнять записями базу данных можно, используя в качестве источника данных выражения, массивы, переменные, константы, результат запроса.

Для дополнения записей в таблицу БД используется команда INSERT, которая предварительно открывает закрытую таблицу БД. Формат команды INSERT:

INSERT INTO <имя_таблицы> [(<имя_поля1>[, <имя_поля2> …])]

VALUES (<выражение1> [, <выражение2> …])

Команда добавляет записи в конец существующей таблицы, используя выражения, перечисленные после слова VALUES. Выражения заносятся в указанные поля. Если опущены имена полей, выражения будут записываться в последовательные поля таблицы данных в соответствии с ее структурой.

В качестве выражений могут использоваться значения переменных или элементов массивов переменных. Например, в СУБД Visual FoxPro для дополнения записей в таблицу данных с использованием массивов используется следующая команда SQL:

INSERT INTO <имя_таблицы>

FROM ARRAY <имя_массива>

Команда добавляет записи в конец таблицы данных, используя данные, содержащиеся в указанном массиве. Данные из массива заносятся последовательно в поля, начиная с первого. Типы соответствующих полей и элементов массива должны совпадать.

Во многих СУБД (но не во всех) SQL-команду INSERT можно использовать с подзапросом - вложенной командой SELECT, если множество дополняемых данных является результатом запроса:

INSERT INTO <имя_таблицы> [(<имя_поля1> [, <имя_поля2> ...])] <подзапрос>

Для модификации данных в таблице БД используется команда UPDATE, которая имеет следующий формат:

UPDATE <имя_таблицы> SET <имя_поля1>= <выражение1> [,<имя_поля2> = <выражение2>..] [WHERE <условие>]

Замена значений происходит в записях, удовлетворяющих условию, указанному после опции WHERE. Если эта опция отсутствует, то замена значений происходит во всех записях таблицы.

Для удаления записей в SQL используется команда DELETE, имеющая следующий формат:

DELETE FROM <имя_таблицы> [WHERE <условия>]

Удаляются записи, удовлетворяющие условию, указанному после опции WHERE. Если эта опция отсутствует, то удаляются все записи из таблицы.

В Visual FoxPro поддерживается команда SQL - DELETE, которая помечает записи на удаление без физического удаления записей. Физическое удаление записей можно реализовать командой PACK.

Контрольные вопросы:

1) К какому классу языковых средств СУБД относится язык SQL?

2) Перечислите категории команд SQL.

3) Перечислите основные типы данных, используемые в SQL.

4) Какая команда SQL используется для определения структуры таблицы данных?

5) Какая команда SQL позволяет добавлять записи в базу данных?

Лекция №11. Запросы SQL

Все запросы на получение практически любого количества данных из одной или нескольких таблиц выполняются с помощью единственного предложения SELECT. В общем случае результатом реализации предложения SELECT является другая таблица. К этой новой (рабочей) таблице может быть снова применена операция SELECT и т.д., т.е. такие операции могут быть вложены друг в друга. Предложение SELECT может использоваться как:

- самостоятельная команда на получение и вывод строк таблицы, сформированной из столбцов и строк одной или нескольких таблиц (представлений);
- элемент WHERE- или HAVING-условия (сокращенный вариант предложения, называемый «вложенный запрос»);
- фраза выбора в командах CREATE VIEW, DECLARE CURSOR или INSERT;
- средство присвоения глобальным переменным значений из строк сформированной таблицы (INTO-фраза).

Здесь в синтаксических конструкциях используются следующие обозначения:

- звездочка (*) для обозначения «все» - употребляется в обычном для программирования смысле, т.е. «все случаи, удовлетворяющие определению»;
- квадратные скобки ([]) означают, что конструкции, заключенные в эти скобки, являются необязательными (т.е. могут быть опущены);
- фигурные скобки ({}) – означают, что конструкции, заключенные в эти скобки, должны рассматриваться как целые синтаксические единицы, т.е. они позволяют уточнить порядок разбора синтаксических конструкций, заменяя обычные скобки, используемые в синтаксисе SQL;
- многоточие (...) указывает на то, что непосредственно предшествующая ему синтаксическая единица факультативно может повторяться один или более раз;
- прямая черта (|) означает наличие выбора из двух или более возможностей. Например, обозначение ASC|DESC указывает, можно выбрать один из терминов ASC или DESC; когда же один из элементов выбора заключен в квадратные скобки, то это означает, что он выбирается, по умолчанию (так, [ASC]|DESC означает, что отсутствие всей этой конструкции будет восприниматься как выбор ASC);
- точка с запятой (;) завершающий элемент предложений SQL;
- запятая (,) используется для разделения элементов списков;

- пробелы ( ) могут вводиться для повышения наглядности между любыми синтаксическими конструкциями предложений SQL;

- прописные латинские буквы и символы используются для написания конструкций языка SQL и должны (если это специально не оговорено) записываться в точности так, как показано;

- строчные буквы используются для написания конструкций, которые должны заменяться конкретными значениями, выбранными пользователем, причем для определенности отдельные слова этих конструкций связываются между собой символом подчеркивания (_);

- термины таблица, столбец, ... заменяют (с целью сокращения текста синтаксических конструкций) термины имя_таблицы, имя_столбца, ..., соответственно;

- термин «таблица» используется для обобщения таких видов таблиц, как базовая_таблица, представление или псевдоним; здесь псевдоним служит для временного (на момент выполнения запроса) переименования и (или) создания рабочей копии базовой_таблицы (представления).

Предложение SELECT (выбрать) имеет следующий формат:

SELECT [DISTINCT] [<псевдоним>.]<выражение> [AS <колонка>][,
[<псевдоним>]<выражение> [AS <колонка>]..]
FROM<имя_таблицы1>[<псевдоним1>][,<имя_таблицы2>
[<псевдоним2>].]
[[INTO <получатель>]/[TO FILE < имя_файла> [ADDITIVE]/TO PRINTER]]
[NOCONSOLE] [PLAIN] [NOWAIT]
[WHERE <условие_связи> [AND <условие_связи>]
[AND/OR <условие_связи >]]
[GROUP BY <колонка> [, <колонка>...]] [HAVING <условие_отбора>]
[ORDER BY <колонка> [ASC/DESC] [,<колонка> [ASC/DESC]...]]

Ниже указывается назначение опций команды.

SELECT - (выбрать) данные из указанных столбцов и (если необходимо) выполнить перед выводом их преобразование в соответствии с указанными выражениями и (или) функциями.

DISTINCT исключает возможность вывода одинаковых строк в выборке.

FROM (из) - перечисленных таблиц, в которых расположены эти столбцы.

WHERE (где) - строки из указанных таблиц должны удовлетворять указанному перечню условий отбора строк.

GROUP BY - (группируя по) указанному перечню столбцов с тем, чтобы получить для каждой группы единственное агрегированное значение, используя во фразе SELECT SQL-функции SUM (сумма), COUNT (количество), MIN (минимальное значение), MAX (максимальное значение) или AVG (среднее значение).

HAVING - (имея) в результате лишь те группы, которые удовлетворяют указанному перечню условий отбора групп.

Кроме традиционных операторов сравнения (= | <> | < | <= | > | >=), в WHERE фразе используются условия BETWEEN (между), LIKE (похоже на), IN (принадлежит), IS NULL (не определено) и EXISTS (существует), которые могут предваряться оператором NOT (не). Критерий отбора строк формируется из одного или нескольких условий, соединенных логическими операторами:

AND - когда должны удовлетворяться оба разделяемых с помощью AND условия.

OR - когда должно удовлетворяться одно из разделяемых с помощью OR условий.

AND NOT - когда должно удовлетворяться первое условие и не должно второе.

OR NOT - когда или должно удовлетворяться первое условие или не должно удовлетворяться второе, причем существует приоритет AND над OR (сначала выполняются все операции AND и только после этого операции OR). Для получения желаемого результата WHERE условия должны быть введены в правильном порядке, который можно организовать введением скобок.

При обработке условия числа сравниваются алгебраически - отрицательные числа считаются меньшими, чем положительные, независимо от их абсолютной величины. Строки символов сравниваются в соответствии с их представлением в коде, используемом в конкретной СУБД, например, в коде ASCII. Если сравниваются две строки символов, имеющих разные длины, более короткая строка дополняется справа пробелами для того, чтобы они имели одинаковую длину перед осуществлением сравнения.

Фраза SELECT может включать не только выражения, но и отдельные числовые или текстовые константы. Следует отметить, что текстовые константы должны заключаться в апострофы (').

В синтаксисе фразы WHERE показано, что для отбора нужных строк таблицы можно использовать операторы сравнения = (равно), <> (не равно), < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), которые могут предваряться оператором NOT, создавая, например, отношения «не меньше» и «не больше».

Для упорядочения по заданной колонке или колонкам используется опция ORDER BY. По умолчанию, сортировка выполняется по возрастанию (ASC), но может быть задана и по убыванию (DESC).

Сокращенный вариант команды SELECT с использованием опции ORDER BY выглядит следующим образом:

SELECT <выражение> FROM <имя_таблицы>

ORDER BY <колонка1> [ASC/DESC] [,<колонка2>..]

Указание колонки упорядочения может выполняться именем или номером колонки.

В опции ORDER BY обычно нельзя использовать вычисляемые выражения. В случае необходимости упорядочения по колонке с вычисляемыми значениями указывается номер колонки.

Для указания объекта получателя данных выборки используется опция INTO или TO.

Ниже приведен сокращенный вариант команды SELECT с опцией INTO/TO, используемый в Visual FoxPro:

SELECT <выражение> FROM <имя_таблицы>

[INTO TABLE <имя_таблицы>] / [INTO CURSOR <имя_курсора>] /

[INTO ARRAY <имя_массива>] /

[TO FILE <имя_файла> [ADDITIVE]] /

[TO PRINTER] [NOCONSOLE] [PLAIN] [NOWAIT]

Типы возможных получателей данных выборки в Visual FoxPro описаны ниже.

TABLE <имя_таблицы> - получателем является новая таблица с указанным именем.

CURSOR <имя_курсора> - результат запроса помещается в курсор с указанным именем. Курсор - это временный набор данных, который может быть областью памяти или временным файлом и имеет режим «только чтение». Данные курсора могут быть, например, предъявлены в команде BROWSE, напечатаны, из них может быть образовано меню и т.д. Курсор может быть обработан другой командой SELECT. К колонкам курсора надо обращаться по имени этих колонок с префиксом - именем курсора (через точку).

ARRAY <имя_массива> - в качестве получателя результата запроса будет использован новый двумерный массив с указанным именем.

Кроме того, данные выборки можно переслать в файл или на принтер. Для этого в команде указывается получатель TO FILE <имя_файла> [ADDITIVE] / TO PRINTER и выборка посылается в текстовый файл с указанным именем или на принтер. Если используется слово ADDITIVE, то выборка будет добавлена в конец существующего файла без его перезаписи.

Контрольные вопросы:

1) Перечислите варианты использования предложения SELECT.

2) Какая опция предложения SELECT используется для упорядочения по заданной колонке (колонкам)?

3) Каким образом в предложении SELECT задать условие для отбора строк, какие операторы при этом используются?

4) Какая опция предложения SELECT используется для указания объекта получателя данных выборки?

5) Назначение операторов SQL: HAVING, DISTINCT, ORDER BY.

Лекция №12. Сложные запросы SQL

В SQL существует ряд специальных стандартных функций (агрегатные функции SQL). Кроме специального случая COUNT(*), каждая из этих

функций оперирует совокупностью значений столбца некоторой таблицы и создает единственное значение, определяемое так:

- COUNT - число значений в столбце;
- SUM - сумма значений в столбце;
- AVG - среднее значение в столбце;
- MAX - самое большое значение в столбце.

Агрегатные функции используются подобно именам полей в операторе SELECT, но с одним исключением: они берут имя поля как аргумент. Это, конечно, отличается от выбора поля, поскольку всегда возвращается одиночное значение, независимо от того, сколько строк находится в таблице. Аргументом агрегатных функций могут быть отдельные столбцы таблиц. Но для того, чтобы вычислить, например, количество различных значений некоторого столбца в группе, необходимо применить ключевое слово DISTINCT совместно с именем столбца.

Например, для вывода минимального, максимального и среднего значений зарплаты (поле ZARP) из таблицы KADR необходимо использовать команду:

SELECT MIN(ZARP), MAX(ZARP), AVG(ZARP) FROM KADR

Нельзя использовать агрегатные функции в предложении WHERE, потому что предикаты оцениваются в терминах одиночной строки, а агрегатные функции - в терминах групп строк.

С функциями SUM и AVG могут использоваться только числовые поля. С функциями COUNT, MAX и MIN могут использоваться как числовые, так и символьные поля. При использовании с символьными полями MAX и MIN будут транслировать их в эквивалент ASCII кода и обрабатывать в алфавитном порядке. Некоторые СУБД позволяют использовать вложенные агрегаты, но это является отклонением от стандарта ANSI со всеми вытекающими отсюда последствиями.

Опция GROUP BY команды SELECT позволяет сгруппировать записи с одинаковым значением указанной колонки (или колонок):

SELECT <выражение> FROM <имя_таблицы>

GROUP BY <колонка1> [,<колонка2>...] [HAVING <условие_отбора>]

Опция GROUP BY задает колонки, по которым производится группирование выходных данных. Все записи таблицы, для которых значения колонок совпадают, отображаются в выборке единственной строкой. Группирование удобно для получения некоторых сводных характеристик группы (суммы, среднего значения, количества записей в группе и т.д.). Предложение GROUP BY позволяет определять подмножество значений в особом поле в терминах другого поля и применять функцию агрегата к подмножеству. Это дает возможность объединять поля и агрегатные функции в едином предложении SELECT.

Внутри групп можно создавать подгруппы.

Опция HAVING <условие_отбора> задает критерий отбора данных в каждую сформированную в процессе выборки группу, т.е. выполняет роль

опции WHERE, но для группируемых данных. Предложение HAVING определяет критерии, используемые, чтобы удалять определенные группы из вывода, точно так же, как предложение WHERE делает это для индивидуальных строк.

Результатом выполнения раздела HAVING является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть TRUE. В частности, если раздел HAVING присутствует в табличном выражении, не содержащем GROUP BY, то результатом его выполнения будет либо пустая таблица, либо результат выполнения предыдущих разделов табличного выражения, рассматриваемый как одна группа без столбцов группирования.

При группировании данных в опции HAVING можно использовать агрегатные функции SQL.

Таким образом, агрегатные функции могут применяться как в выражении вывода результатов строки SELECT, так и в выражении условия обработки сформированных групп HAVING. В этом случае каждая агрегатная функция вычисляется для каждой выделенной группы. Значения, полученные при вычислении агрегатных функций, могут быть использованы для вывода соответствующих результатов или для условия отбора групп.

В результат можно включить значение поля группировки и несколько агрегатных функций, а в условиях группировки можно использовать несколько полей. При этом группы образуются по набору заданных полей группировки. Операции с агрегатными функциями могут быть применены к объединению множества исходных таблиц.

Команду SELECT можно использовать для одновременной выборки данных из нескольких таблиц.

Сокращенный вариант предложения SELECT (выбрать) в этом случае имеет следующий формат:

SELECT [<псевдоним>.]<выражение> [AS <колонка>][,

[<псевдоним>]<выражение> [AS <колонка>]..]

FROM<имя_таблицы1>[<псевдоним1>][,<имя_таблицы2>
[<псевдоним2>].]

[WHERE <условие_связи> [AND <условие_связи>]

[AND/OR <условие_связи >]] [AND <условие_отбора>]

[AND/OR <условие_отбора>]]

Если имена полей, выбираемых из разных таблиц совпадают, то такие колонки получают совпадающие имена, к которым присоединяется одна из букв (по алфавиту), например, FAM_A, FAM_B и т.д. Аналогичным образом даются имена колонкам, полученным в результате вычисления выражений. Их имена состоят из слова EXP и последовательных чисел (EXP_1, EXP_2 и т.д.). Исключения составляют выражения, использующие собственные функции SQL: AVG, MIN, MAX, SUM, COUNT. Последняя функция может иметь в качестве аргумента звездочку (COUNT(*)), что означает подсчет всех записей, попавших в выборку. Имена колонок в этом случае будут включать имена

функций. Вместо имен, формируемых по умолчанию, можно назначить колонкам другие имена, указав их после слова AS в виде <выражение> AS <новое_имя_колонки>.

Псевдонимом может быть не только официальный псевдоним (ALIAS) таблицы данных, но и любое другое имя, которое присваивается в команде SELECT. Это задаваемое временное имя указывается в опции <псевдоним> после слова FROM за именем таблицы и используется только в данной команде SELECT в других ее опциях (локальный псевдоним).

Например, для вывода всех фамилий из двух таблиц данных KADR и STUD с использованием локальных псевдонимов P и T, а также с заменой при выводе имен колонок FAM_A и FAM_B на KFIO и STFIO соответственно, используется команда:

SELECT P.FAM AS KFIO, T.FAM AS STFIO FROM KADR P, STUD T

В команде SELECT можно задать достаточно сложные условия для выборки данных в запрос. Условие связи применяется в случае, если выборка делается более, чем из одной таблицы данных, и определяет критерий объединения данных из разных таблиц. В условии связи указываются поля из разных таблиц с псевдонимами и используются знаки отношений =, #, = =, >, >=, <, <=. Допускается задание нескольких критериев, соединенных знаком AND.

Например, для вывода на экран из таблицы KADR фамилий сотрудников (поле FAM), а из таблицы TABEL - соответствующего количества рабочих дней (поле WD) для записей, у которых совпадают табельные номера, необходимо использовать команду с условием связи (в качестве псевдонимов таблиц указаны имена таблиц):

SELECT KADR.FAM, TABEL.WD FROM KADR, TABEL WHERE KADR.TAB=TABEL.TAB

Условие выборки строится аналогично, но из выражений только для одной таблицы, и допускается использование логических операторов OR, AND и NOT.

Условия могут содержать следующие операторы SQL: LIKE, BETWEEN, IN. Эти операторы можно комбинировать с помощью связок OR, AND, NOT и скобок.

Оператор LIKE позволяет построить условие сравнения по шаблону, где символ «_» указывает единичный неопределенный символ в строке, а символ «%» - любое их количество. Формат оператора LIKE:

<выражение> LIKE <шаблон>.

Оператор BETWEEN задает начальное и конечное значение диапазона и проверяет, находится ли выражение, стоящее слева от оператора, в указанном диапазоне. Формат оператора BETWEEN:

<выражение> BETWEEN <нижнее значение> AND <верхнее значение>.

Оператор IN проверяет, находится ли выражение, стоящее слева от слова IN, среди перечисленных справа от него. Формат оператора IN:

<выражение> IN (<выражение1>, <выражение2>,...)

Оператор JOIN позволяет соединить данные из нескольких таблиц, связав таблицы по какому-либо критерию (условию). Возможны четыре различные типы связей.

INNER JOIN – эта связь позволяет включить в выборку только те значения из первой и второй таблицы, которые удовлетворяют указанному условию.

LEFT OUTER JOIN – эта связь позволяет включить в выборку все записи из первой таблицы и записи из второй таблицы, удовлетворяющие указанному условию.

RIGHT OUTER JOIN – эта связь позволяет включить в выборку все записи из второй таблицы и записи из первой таблицы, удовлетворяющие указанному условию.

FULL OUTER JOIN – эта связь позволяет включить в выборку все записи из первой и второй таблицы.

В этом случае действие, производимое оператором SELECT при выборке из двух таблиц, называют слиянием таблиц, а если производится слияние с OUTER, то такое слияние таблиц называется внешним. Внутри одного запроса можно использовать несколько таблиц с OUTER (такие таблицы называют подчиненными, а таблицы без OUTER - ведущими).

Например, для вывода на экран из таблицы KADR фамилий всех сотрудников (поле FAM), а из таблицы TABEL - соответствующего количества рабочих дней (поле WD) для записей, у которых совпадают табельные номера, необходимо использовать команду с условием связи LEFT OUTER JOIN (в качестве псевдонимов таблиц указаны имена таблиц):

```
SELECT KADR.FAM, TABEL.WD FROM KADR LEFT OUTER JOIN
TABEL
ON KADR.TAB=TABEL.TAB
```

При выполнении этой команды последовательно будут перебираться все записи из таблицы KADR (левой таблицы) и включаться в результат запроса. Для каждой записи из KADR будет просмотрена таблица TABEL. Как только будет найдена запись из TABEL, удовлетворяющая условию KADR.TAB=TABEL.TAB, в результат для имеющейся строки с соответствующим значением табельного номера добавится значение поля WD соответствующей записи из TABEL. Если же в TABEL не будет найдено ни одной записи, поле WD получит значение NULL.

Контрольные вопросы:

1) Перечислите специальные стандартные функции (SQL-функции) и их назначение.

2) Какая опция команды SELECT позволяет сгруппировать записи с одинаковым значением указанной колонки (колонок)?

3) Каким образом команда SELECT используется для одновременной выборки данных из нескольких таблиц?

4) Какая опция команды SELECT задает критерий отбора данных в каждую сформированную в процессе выборки группу.

5) Назначение операторов SQL: LIKE, BETWEEN, IN.

Лекция №13. Архитектура клиент-сервер в технологии БД

Классификацию СБД можно проводить по многочисленным критериям, ниже приведена классификация по некоторым из них.

По числу уровней в архитектуре выделяют системы БД одноуровневые, двухуровневые, трёхуровневые и многоуровневые. Трехуровневая архитектура СБД является стандартной, но необязательной.

В зависимости от типа схемы данных реализуемой СУБД, системы БД делятся на форматированные (структурированные) и неструктурированные.

В структурированных СБД используются детерминированные схемы данных: типы объектов, их свойства, их взаимосвязи заранее определены. В свою очередь, структурированные СБД по типу используемой модели данных делятся на реляционные, сетевые, иерархические.

В неструктурированных СБД совокупность объектов, их свойств и взаимосвязей определяется в момент появления объектов в поле зрения СУБД. В неструктурированных системах условно можно выделить модели дескрипторные, дескрипторные с грамматикой, фреймовые модели, модели на семантических сетях.

По используемому языку общения различают СБД замкнутые и открытые. В открытых системах для программирования используется базовый включающий язык, дополненный операторами языка манипулирования данными. В замкнутых системах используется неалгоритмический язык высокого уровня, близкий к естественному языку.

В зависимости от сферы применения системы БД могут быть разделены на универсальные, проблемно-ориентированные и с широкой областью применения, которые наиболее распространены. Эта классификация связана с типом объектов, хранимых в базе данных. В универсальных СБД могут интегрироваться как традиционные, так и нетрадиционные данные (текст, звук, рисунок, видео, страницы HTML и др.). Проблемная ориентация проблемно-ориентированных СБД обуславливается разными причинами. Например, особенностями типов объектов только из определенной предметной области; особенностями используемых языковых средств; включением в СУБД специальных процедур обработки данных и т.д.

В зависимости от типов объектов (типа данных), хранимых в БД, СБД, подразделяются на: административные (хранятся символьные, числовые, логические данные, даты); графические (хранятся графические данные); библиографические или полнотекстовые (хранятся тексты в целом: статьи, журналы и т.д.); комбинированные (все типы данных).

По характеру организации хранения и обработки данных СБД делятся на локальные (на одной ЭВМ) и распределённые (на нескольких ЭВМ,

работающих в локальной или глобальной сети). В первом случае говорят о доступе к локальным данным, во втором - о доступе к удаленным данным.

Локальные данные, как правило, располагаются на жестком диске компьютера, на котором работает пользователь, и находятся в монопольном ведении этого пользователя. Пользователь при этом работает автономно, не зависит от других пользователей и никоим образом не влияет на их работу. Удаленные данные располагаются вне компьютера пользователя (пользователей) на специально выделенном для этих целей компьютере.

Распределённые СБД в свою очередь могут быть однородными или разнородными. В однородных РСБД используются СУБД и ЭВМ одного типа, а в разнородных - различного типа. Для распределённых СБД в сети по её узлам распределяются либо БД, либо БД и СУБД одновременно.

В распределённых СБД в настоящее время используются две технологии (архитектуры) обработки удаленных данных: файл-серверная и клиент–серверная.

Архитектура «файл-сервер». В стандартной файл-серверной архитектуре данные располагаются на файл-сервере, который является, по сути, пассивным источником данных для рабочих станций. Файл-сервером называется компьютер, используемый как централизованное хранилище коллективно используемых файлов. Рабочие станции - это компьютеры любого класса, совместно использующие ресурсы серверов при решении своих прикладных задач.

Вся ответственность за получение и обработку данных, а также за поддержание целостности базы данных лежит на приложении, запущенном с рабочей станции. При этом, поскольку обработка данных осуществляется на рабочей станции, по сети передается вся необходимая для этой обработки информация, хотя интересующий пользователя объем данных может быть намного меньше пересылаемого. Например, если пользователя интересуют сотрудники какого-либо предприятия, участвующие в конкретном проекте, его приложение получит данные о всех сотрудниках и всех проектах из базы данных, и только после этого произведет требуемую выборку.

Исторически на персональных компьютерах использовался именно этот подход как более простой в освоении. Однако большой объем передаваемых по сети данных существенно ограничивает число пользователей в сети. Этот основной и самый существенный недостаток заставил искать способы уменьшения нагрузки на сеть.

Архитектура «клиент-сервер» предназначена для разрешения проблем файл-серверных приложений путем разделения компонентов приложения и размещения их там, где они будут функционировать более эффективно. Особенностью архитектуры клиент-сервер является использование выделенных серверов баз данных, понимающих запросы и выполняющих поиск, сортировку и агрегирование информации на месте без излишней перекачки данных на рабочие станции.

Архитектура «клиент-сервер». В архитектуре «клиент-сервер» для обработки данных выделяется специальное ядро – так называемый сервер баз данных, который принимает на себя функции обработки запросов пользователей, именуемых теперь клиентами.

Сервер баз данных представляет собой программу, выполняющуюся, как правило, на мощном компьютере. «Приложения-клиенты» посылают с рабочих станций запросы на выборку (вставку, обновление, удаление) данных. При этом сервер выполняет всю работу по отбору данных, отправляя клиенту только требуемые данные, которые, в общем случае, составляют малую часть от общего объема БД. Поэтому в сети не наблюдается резкого увеличения нагрузки при увеличении числа клиентов. Клиентские же приложения могут выполняться на менее мощных (по сравнению с сервером) компьютерах.

Если приведенный выше пример реализовать на клиент-серверной архитектуре, то «приложение – клиент» получит от сервера в качестве результата данные только о тех сотрудниках, которые участвуют в указанном проекте.

Другая отличительная черта серверов БД - наличие словаря данных, в котором записаны структура БД, ограничения целостности данных, форматы и даже серверные процедуры обработки данных по вызову или по событиям в программе. Объектами разработки в таких приложениях, помимо диалога и логики обработки, являются прежде всего реляционная модель данных и связанный с ней набор SQL-операторов для типовых запросов для этой БД.

Наиболее важным результатом перехода в архитектуру «клиент-сервер» является гарантированное сохранение логической целостности базы данных, т.е. система становится более устойчивой и более защищенной. Достигается это благодаря возможности переложить заботу о сохранении целостности базы данных на сервер. Для этого серверы обладают большим набором встроенных механизмов, защищающих систему от неверных действий клиентов. Среди этих механизмов можно назвать такие, как ограничение целостности, декларативная ссылочная целостность, триггеры, виртуальные таблицы (представления), авторизация пользователей и др.

Таким образом, клиент-серверная архитектура обеспечивает решение трёх важных задач: уменьшение нагрузки на сеть, уменьшение требований к компьютерам клиента, повышение надёжности и сохранение логической целостности базы данных.

Большинство конфигураций «клиент-сервер» использует двухзвенную модель, состоящую из клиента, который обращается к услугам сервера. Для эффективной реализации такой схемы часто применяют неоднородную сеть. Типовое определение архитектуры «клиент-сервер»: приложение на клиенте, СУБД - на сервере.

Поскольку эта схема предъявляет наименьшие требования к серверу, она обладает наилучшей масштабируемостью. Однако сложные приложения, вызывающие большое взаимодействие с БД, могут жестко загрузить как клиента, так и сеть. Результаты SQL-запроса должны вернуться клиенту для

обработки, потому что там находится логика принятия решения. Такая схема возлагает дополнительное бремя администрирования приложений, разбросанных по различным клиентским узлам.

Можно сократить нагрузку на клиента и сеть, переместив обработку на сервер приложений. Переместив с клиента часть логики приложения на сервер приложений, получим систему «клиент-сервер» с разделенной логикой (двухзвенная модель).

Приложения баз данных представляют собой информационные системы различного типа, ориентированные на конечного пользователя. Они должны быть удобны в эксплуатации и иметь дружественный комфортный пользовательский интерфейс. Обычно приложения баз данных создаются для специалистов, не обладающих никакими знаниями в сфере программирования и знакомых с работой компьютера лишь поверхностно на уровне оператора. Простота, удобство, надежность, сохранение целостности данных, учет требований и пожеланий пользователя в соответствии с поставленными задачами, а также легкость восприятия информации - это основные требования при создании приложения. В связи с этим, можно сформулировать ряд требований к разрабатываемому приложению, изложенные ниже.

Автономность эксплуатации. При эксплуатации система не должна быть связанной ни с какими другими программными средствами.

Функциональность. Приложение должно полностью соответствовать своему функциональному назначению.

Экономичность. Приложение должно экономно использовать вычислительные ресурсы для обеспечения его работы на небольших, экономичных конфигурациях аппаратного обеспечения без потери функциональности.

Простота эксплуатации. При работе с приложением не должно требоваться специальных знаний по программированию, должны отсутствовать требования к профессиональной подготовке пользователя в области вычислительной техники. Достаточно, чтобы пользователь мог запустить компьютер, работать с клавиатурой, совершать некоторые действия, такие, как установка дискеты, копирование информации и т.д.

Дружественность интерфейса. Работа приложения должна сопровождаться подсказками, ориентирующими пользователя на выполнение определенных действий. В приложении должна быть предусмотрена возможность контроля за действиями пользователя, диагностика ошибочных действий и выдача на экран рекомендаций по их устранению.

Комфортность интерфейса. Отображаемая информация должна быть структурированной для обеспечения комфортных условий взаимодействия в диалоговой системе и согласованной с возможностями человека по восприятию и обработке данных.

Надежность и защита данных. Должно быть обеспечено сохранение целостности и непротиворечивости данных, ограничение несанкционированного доступа к ним.

Современные визуальные технологии обеспечивают быструю разработку приложений (RAD - Rapid Application Designer), отвечающих вышеперечисленным требованиям.

Среды разработки приложений для серверов баз данных.

Среды разработки приложений для серверов БД представляют собой системы программирования четвертого поколения 4GL, инструментальные средства быстрой разработки приложений RAD (Rapid Application Development).

Особенностями этой подгруппы средств являются: реализация удаленного доступа к СУБД по двухзвенной схеме «клиент-сервер»; связь клиентских приложений с серверами БД с помощью непроцедурного языка структурированных запросов SQL; обеспечение целостности БД, включая целостность транзакций; поддержка хранимых процедур на серверах БД, реализация клиентских и серверных триггеров-процедур; генерация элементов диалогового интерфейса и отчетов.

Любая инструментальная среда разработки приложений, поддерживающая язык структурированных запросов SQL, может быть использована для разработки клиент-серверных приложений: инструментальные средства клиентских СУБД, инструментальные средства полнофункциональных СУБД, универсальные инструментальные среды программирования и т.д. В качестве примера можно назвать инструменты Informix/4GL, Oracle*Forms и др.

Сейчас новые среды разработки SQL-серверов БД (Informix NewEra и Oracle Power Objects) развиваются в сторону независимых от СУБД инструментов.

Независимые инструментальные средства, ориентированные на многие платформы СУБД, представлены в виде средств быстрой разработки приложений RAD. Для таких средств создания приложений «клиент-сервер» характерны: возможность распределения приложения на клиентах и/или серверах; создание приложений для разных серверов БД; поддержка спецификации ODBC для доступа к различным серверам БД, включая СУБД для ПК; связь с мониторами транзакций для организации трехзвенной архитектуры приложений клиент-сервер; объектно-ориентированное программирование приложений; визуальный характер генерации приложения; ведение репозитария объектов и их свойств, что облегчает интеграцию со средствами автоматизации проектирования программ CASE; управление проектами и версиями приложений; интеграция приложения с электронной почтой и средствами офисной автоматизации.

Контрольные вопросы:

1) По каким критериям возможна классификация СБД?

2) В чем отличие файл-серверной технологии обработки данных от клиент-серверной технологии?

3) В чем заключается основное преимущество клиент-серверной технологии обработки данных?

4) В чем отличие двухзвенной от трехзвенной модели клиент-серверной технологии?

5) Перечислите требования, предъявляемые к разрабатываемому приложению.

Лекция №14. Объектно-ориентированное программирование (ООП) в СБД

Объектно-ориентированное программирование (ООП) основывается на понятии объекта, в то время как традиционное процедурное программирование основывается на процедурах. Понятие объекта, являющееся основным в ООП, позволяет очень эффективно использовать модульный принцип составления программ и сделало возможным визуальное программирование и проектирование БД и их приложений.

Основная особенность объектно-ориентированного языка программирования состоит в том, что программа организуется по объектам программирования. Объекты содержат инструкции (называемые методами) и данные (называемые свойствами), которые определяют поведение объекта. Метод – это функция или процедура, которая управляет работой объекта. Внешний вид и работа объекта определяются характеристиками, которые являются свойствами объекта. Свойства объекта могут быть изменяемыми и неизменяемыми, но их значения всегда можно прочитать.

Объект можно определить как совокупность программно связанных методов и свойств, выполняющих одну функционально связанную задачу и представляющих собой единое целое. В процессе работы программы объект выполняет свое стандартное поведение и в случае необходимости изменяет данные для отражения влияния назначенного ему действия. Объектом можно назвать любой предмет: окно, поле ввода, кнопку и т.д.

При объектно-ориентированном программировании на первый план выступают данные, несущие информацию об объектах и осуществляющие обмен информацией между объектами. В модели электронной схемы в роли управляющих данных могут быть электрические сигналы, в модели экономики - количество товаров на рынке, в модели многооконной операционной оболочки - все происходящие в системе события. В ООП управляющие данные принято называть сообщениями.

В ООП используется событийная модель управления. Событие представляет собой некоторое действие, которое активизирует стандартную реакцию объекта. Событие обрабатывается методом. В качестве события может рассматриваться нажатие кнопки мыши, выбор пункта меню, открытие таблицы и т.д. Порядок выполнения действий определяется прежде всего событиями, возникающими в системе, и реакцией на них объектов. Ниже

приведена последовательность выполнения действий в системе при условии предварительно запрограммированных реакций системы:

1) В системе возникает событие (нажатие клавиши, изменение значения поля и т.д.).

2) Определяется связанный с этим событием объект (поле таблицы, кнопка, форма и т.д.).

3) Вызывается соответствующий событию метод объекта, который содержит действия по обработке возникшего события.

Помимо ключевого понятия объект, в объектно-ориентированных языках программирования существует и широко применяется понятие класс. Объект можно создать программно или визуально на основе класса. Классы и объекты тесно связаны между собой, но эти понятия не тождественны. Класс содержит информацию о том, как должен выглядеть объект и определяет выполняемые им действия. Объект является экземпляром класса, который наследует характеристики класса.

Выделяют три основные характеристики объектно-ориентированного программирования: наследование, инкапсуляция, полиморфизм.

Наследование. Все объекты создаются на основе классов и наследуют свойства и методы классов. Классы могут, в свою очередь, создаваться на основе других классов. Такие классы называются подклассами. Они наследуют все свойства и методы своих родительских классов. Дополнительно можно определить для подкласса новые методы и свойства. Подклассы позволяют сократить объем программирования и максимально использовать предыдущий опыт работы. Кроме того, изменение свойств и методов родительского класса отслеживается в подклассах, созданных на основе этого класса, а также в объектах, созданных на основе подклассов. Таким образом, можно без особых затрат изменить характеристики всего приложения.

Инкапсуляция. Методы и свойства, объединенные в объекте, не могут существовать вне его. При копировании объект копируется как единое целое. Принцип объединения в единое целое данных и программного кода, описывающего поведенческие свойства объекта, известен под термином инкапсуляция. При этом можно пользоваться как специальным типом данных, введенным в языки программирования для реализации технологии ООП (например, в C++, Simula, Prolog), так и обычными синтаксическими конструкциями.

Полиморфизм. При традиционном процедурном программировании имя вызываемой подпрограммы или функции однозначно определяет выполняемый код. В объектно-ориентированном программировании можно использовать одни и те же имена методов для выполнения совершенно разных действий. Например, метод open() может использоваться как для открытия формы, так и открытия базы данных. Выполняемое действие зависит от типа объекта, к которому применяется данный метод. Такой подход значительно упрощает программирование, позволяя использовать одинаковые смысловые наименования для выполнения разнообразных действий.

Классом называют шаблон, который описывает методы и свойства, используемые для определённого типа объектов. Классы имеют иерархическую структуру. На основе класса системы (базового класса) можно создать класс разработчика приложений, который, в свою очередь, может быть родительским классом для подклассов пользователя. Классы как базовые, так и созданные, хранятся в библиотеках классов. При создании объектов можно использовать базовые классы системы, а также созданные новые специальные классы и подклассы.

Существуют разные типы классов. Большинство базовых классов являются видимыми, некоторые классы используются для объединения объектов и не отображаются в форме. Кроме того, одни классы допускают вложение других классов, другие - не допускают. Можно использовать не только базовые классы, но и создавать собственные, Ниже дается краткая характеристика основным типам классов.

Классы - элементы управления, позволяют создавать объекты, с которыми можно обращаться только как с единым целым (линия, кнопка, список, таймер и другие элементы управления).

Классы - контейнеры, позволяют создавать объекты, которые могут содержать внутри себя другие объекты, позволяя к тому же манипулировать этими внутренними объектами. Например, контейнер - таблица содержит столбцы и заголовки столбцов, контейнер - форма содержит различные элементы управления.

По событиям, и в полной мере соответствуют требованиям, предъявляемым к средствам проектирования приложений. Можно перечислить все элементы интерфейса современных СУБД и большинство из них окажутся объектами. Специальные инструментальные средства СУБД позволяют визуально проектировать базы данных и другие компоненты приложений баз данных (конструкторы, генераторы, редакторы, мастера, построители и т.д).

Среды программирования современных СУБД предоставляют.

Различают визуальные классы - видимые на экране и составляющие основу пользовательского интерфейса (например, кнопка, таблица, форма и т.д.) и не визуальные, которые видны только в процессе проектирования (например, таймер).

Едва ли можно найти предметную область, объекты которой нельзя было бы описать в терминах иерархических структур или классификаций. Классификации удобны тем, что позволяют разнести свойства объектов по иерархическим уровням, т.е. структурировать их. При этом реализуется механизм наследования - свойства старшего класса автоматически переносятся на дочерний уровень.

В объектно-ориентированных языках программирования, где механизм наследования реализуется при помощи специальных синтаксических конструкций, дочерние классы автоматически наследуют все свойства объекта, в том числе и функции, описывающие его поведение.

СУБД состоит из отдельных объектов (компонентов), которые используются для хранения информации, ее отображения и редактирования.

В СБД все данные хранятся в базе данных, которая состоит из таблиц, отношений между таблицами, индексов, триггеров, хранимых процедур и т.д.

Для отображения и редактирования данных используются формы, отчеты, запросы и программы. Чтобы создавать формы, отчеты и запросы, приме-няются мастера и конструкторы. Формы и отчеты являются составными объек-тами, так как они состоят из более мелких объектов (таких, как поля, кнопки, диаграммы, рамки, OLE-компоненты и т. п.), которые называются объектами интерфейса.

Для создания объектов на основе классов применяют специальные средства визуального программирования. Визуальное программирование донесло основные положения объектно-ориентированного программирования до конечного пользователя. Вместо того, чтобы писать многие строки кодов, определяющие поведение объекта, пользуются прототипами необходимых объектов - классами. Средства визуального программирования предоставляют соответствующие классы для создаваемых объектов, которые можно настроить по своему усмотрению. Например, практически все визуально программируемые языки содержат прототипы кнопки, которая утопает и всплывает при нажатии на нее пользователем. Это дает возможность не писать многие строки кода для создания кнопки, а воспользоваться уже готовым прототипом кнопки.

Языки программирования современных СУБД являются объектно-ориентированными, визуально программируемыми языками, управляемыми в распоряжение пользователя много самых разнообразных панелей инструментов для работы с базами данных, формами, отчетами и запросами. Эти панели инструментов содержат набор кнопок, причем этот набор зависит от назначения конкретной панели инструментов.

Программы, написанные на языке программирования СУБД, являются объектно-ориентированными. С их помощью обрабатываются события в форме, создаются объекты, изменяются их свойства, осуществляются различные вычисления, выполняется управление базой данных.

Для удобства работы можно объединить программы в библиотеки.

Контрольные вопросы:

1) Какое понятие является основным в объектно-ориентированном программировании?

2) Что собой представляют метод и свойство объекта?

3) Какая модель управления используется в объектно-ориентированном программировании?

4) Что собой представляет событие?

5) Какие характеристики присущи объектно-ориентированному программированию?

6) Какое определение можно дать объекту?

7) Что означает понятие класса в объектно-ориентированном программировании?

8) Какие существуют основные типы классов?

9) В чем отличие класса- контейнера от класса-элемента управления?

Лекция №15. Защита баз данных. Целостность и сохранность баз данных

Термин безопасность относится к защите данных от несанкционированного доступа, изменения или разрушения данных, а целостность – к точности или истинности данных.

Как при обеспечении безопасности, так и при обеспечении целостности система вынуждена проверить, не нарушают ли выполняемые пользователем действия некоторых правил. Эти правила должны быть заданы (обычно администратором базы данных) и сохранены в системном каталоге.

Среди многочисленных аспектов проблемы безопасности необходимо отметить следующие:

- правовые, общественные и этические аспекты (имеет ли право некоторое лицо получить запрашиваемую информацию, например об оценках студента);
- физические условия (например, закрыт ли данный компьютер или терминальная комната или защищен каким-либо другим образом);
- организационные вопросы (например, как в рамках предприятия, обладающего некой системой, организован доступ к данным);
- вопросы реализации управления (например, если используется метод доступа по паролю, то как организована реализация управления и как часто меняются пароли);
- аппаратное обеспечение (обеспечиваются ли меры безопасности на аппаратном уровне, например, с помощью защитных ключей или привилегированного режима управления);
- безопасность операционной системы (например, затирает ли базовая операционная система содержание структуры хранения и файлов с данными при прекращении работы с ними).

Методы обеспечения безопасности.

В современных СУБД поддерживается один из двух широко распространенных подходов к вопросу обеспечения безопасности данных, а именно избирательный подход или обязательный подход.

В случае избирательного управления пользователь обладает различными правами (привилегиями или полномочиями) при работе с разными объектами.

В случае обязательного управления, наоборот, каждому объекту данных присваивается некоторый классификационный уровень, а каждый пользователь обладает некоторым уровнем допуска.

Для того чтобы разобраться, какие правила безопасности к каким запросам доступа применяются, в системе должны быть предусмотрены

способы опознания источника этого запроса, т.е. опознания запрашивающего пользователя. Поэтому в момент входа в систему от пользователя обычно требуется ввести не только его идентификатор (например, имя или должность), но также и пароль (чтобы подтвердить свои права на заявленные ранее идентификационные данные).

Избирательное управление доступом.

Избирательное управление доступом поддерживается многими СУБД. Избирательное управление доступом поддерживается в языке SQL.

В общем случае система безопасности таких СУБД базируется на трех компонентах.

Пользователи. СУБД выполняет любое действия с БД от имени какого-то пользователя. Каждому пользователю присваивается идентификатор – короткое имя, однозначно определяющее пользователя в СУБД. Для подтверждения того, что пользователь может работать с введенным идентификатором используется пароль. Таким образом, с помощью идентификатора и пароля производится идентификация и аутентификация пользователя.

Объекты БД. По стандарту SQL2 защищаемыми объектами в БД являются таблицы, представления, домены и определенные пользователем наборы символов. Большинство коммерческих СУБД расширяет список объектов, добавляя в него хранимые процедуры и др. объекты.

Привилегии. Привилегии показывают набор действий, которые возможно производить над тем или иным объектом. Например, пользователь имеет привилегию для просмотра таблицы.

Обязательное управление доступом.

Методы обязательного управления доступом применяются к базам данных, в которых данные имеют достаточно статичную или жесткую структуру, свойственную, например, правительственным или военным организациям. Как уже отмечалось, основная идея заключается в том, что каждый объект данных имеет некоторый уровень классификации, например: секретно, совершенно секретно, для служебного пользования и т.д., а каждый пользователь имеет уровень допуска с такими же градациями, что и в уровне классификации. Тогда на основе этих сведений можно сформулировать два очень простых правила безопасности:

- пользователь имеет доступ к объекту, только если его уровень допуска больше или равен уровню классификации объекта.

- пользователь может модифицировать объекту, только если его уровень допуска равен уровню классификации объекта.

Шифрование данных.

До сих пор подразумевалось, что предполагаемый нелегальный пользователь пытается незаконно проникнуть в базу данных с помощью обычных средств доступа, имеющихся в системе. Теперь следует рассмотреть случай, когда такой пользователь пытается проникнуть в базу данных, минуя систему, т.е. физически перемещая часть базы данных или подключаясь к

коммуникационному каналу. Наиболее эффективным методом борьбы с такими угрозами является шифрование данных, т.е. хранение и передача особо важных данных в зашифрованном виде.

Для обсуждения основных концепций кодирования данных следует ввести некоторые новые понятия. Исходные (незакодированные) данные называются открытым текстом. Открытый текст шифруется с помощью специального алгоритма шифрования. В качестве входных данных для такого алгоритма выступают открытый текст и ключ шифрования, а в качестве выходных – зашифрованная форма открытого текста, которая называется зашифрованным текстом. Если детали алгоритма шифрования могут быть опубликованы или, по крайней мере, могут не утаиваться, то ключ шифрования обязательно хранится в секрете. Именно зашифрованный текст, который непонятен тем, кто не обладает ключом шифрования, хранится в базе данных и передается по коммуникационному каналу.

Контрольный след выполняемых операций.

Не бывает неуязвимых систем безопасности, поскольку настойчивый потенциальный нарушитель всегда сможет найти способ преодоления всех систем контроля. Поэтому при работе с очень важными данными или при выполнении критических операций возникает необходимость регистрации контрольного следа выполняемых операций. Если, например, противоречивость данных приводит к подозрению, что совершено несанкционированное вмешательство в базу данных, то контрольный след должен быть использован для прояснения ситуации и подтверждения того, что все процессы находятся под контролем. Если это не так, то контрольный след поможет обнаружить нарушителя.

Для сохранения контрольного следа обычно используется особый файл, в котором система автоматически записывает все выполненные пользователями операции при работе с обычной базой данных.

Поддержка мер обеспечения безопасности в языке SQL.

В действующем стандарте языка SQL предусматривается поддержка только избирательного управления доступом. Она основана на двух более или менее независимых частях SQL. Одна из них называется механизмом представлений, который (как говорилось выше) может быть использован для скрытия очень важных данных от несанкционированных пользователей. Другая называется подсистемой полномочий и наделяет одних пользователей правом избирательно и динамично задавать различные полномочия другим пользователям, а также отбирать такие полномочия в случае необходимости.

Механизм представлений языка SQL позволяет различными способами разделить базу данных на части таким образом, чтобы некоторая информация была скрыта от пользователей, которые не имеют прав для доступа к ней. Однако этот режим задается не с помощью параметров операций, на основе которых санкционированные пользователи выполняют те или иные действия с заданной частью данных. Эта функция выполняется с помощью директивы GRANT.

Создатель объекта также получает право предоставить привилегии доступа какому-нибудь другому пользователю с помощью оператора GRANT. Ниже приводится синтаксис утверждения GRANT:

GRANT {SELECT|INSERT|DELETE|(UPDATE столбец, …)}, …
ON таблица TO {пользователь | PUBLIC} [WITH GRANT OPTION]

Контрольные вопросы:

1) Что собой представляет защита информации?

2) В чем заключается безопасность данных?

3) Какие существуют основные направления защиты и безопасности информационных систем на основе технологии баз данных?

4) Какими возможностями по организации защиты и безопасности обладают серверы баз данных?

5) Какие существуют аспекты проблемы безопасности данных?

Глоссарий

Предметная область - часть реального мира, в которой можно выделить один или несколько объектов.

Объект - человек, предмет, событие, место, явление, понятие.

Информация - отображение предметной области, существующее в представлении людей.

Данные - отображение предметной области, хранящееся в компьютере.

Взаимосвязь - выражает отображение или связь между двумя множествами данных, бывает трех типов: «один к одному» (1:1), «один ко многим» (1:M или 1: ∞), «многие ко многим» (M : M).

База данных - представляет собой поименованную совокупность данных, отображающую состояние множества объектов из предметной области, их атрибутов (свойств) и взаимоотношений.

Первичный ключ – это атрибут , идентифицирующий объект (запись) уникальным образом.

Вторичный ключ - идентифицирует группу объектов (записей).

Банк данных - организационно-техническая система, представляющая собой совокупность баз данных, технических и программных средств формирования и ведения этих баз и коллектива специалистов, обеспечивающих функционирование этой системы.

Система управления базой данных (СУБД) – совокупность языковых и программных средств, предназначенных для создания и ведения баз данных.

Свойства данных в БД - интеграция, независимость, отсутствие дублирования, защита и целостность.

Целостность данных - безошибочность, точность и достоверность данных в БД в каждый момент времени.

Схема - логическое описание всех хранимых данных.

Подсхема - описание данных, которые используются каким-либо приложением или пользователем.

Администратор базы данных - отвечает за создание и ведение БД.

ЯОД - язык описания данных, используется для логического описания данных.

ЯМД - язык манипулирования данными, позволяет реализовать интерфейс между приложением и СУБД.

ЯЗ - язык запросов пользователя, позволяет выбирать из БД все требуемые данные в соответствии с задаваемыми критериями.

Внешнее представление - совокупность требований к данным некоторого конкретного приложения или пользователя.

Концептуальное представление – это полная совокупность всех требований к данным, полученным из пользовательских представлений о предметной области.

Внутреннее представление – это сама реальная база данных, реализованная в памяти компьютера средствами выбранной СУБД.

Внешняя модель (подсхема) - описание каждого внешнего представления.

Концептуальная модель - реализация концептуального представления, в ней содержится описание объектов, их атрибутов и взаимосвязей (СУБД-независимая схема).

Внутренняя модель – реализует внутреннее представление, включает логическую модель и физическую модель.

Логическая модель – СУБД-ориентированная схема данных.

Физическая модель - специфицирует размещение файлов БД, методы доступа и технику индексирования данных.

Загрузка - первоначальное заполнение БД данными.

Реляционная модель – представляет данные в виде отношение - атрибут, объект представляется в виде отношения, а его свойства – в виде совокупности атрибутов.

Нормализация отношений - процесс построения оптимальной структуры отношений и связей в реляционной базе данных путем избавления от нежелательных зависимостей.

Реляционная алгебра - система операций, используемая для манипулирования отношениями.

Иерархическая модель - представляет данные в виде поле – сегмент – запись, связь между данными является иерархической с четко установленными уровнями вложенности.

Сетевая модель - используется понятия элемент – запись – набор для представления данных, успешно реализует взаимосвязь М : М.

Список литературы

1 Кузнецов С.Д. Основы современных баз данных. Центр ИнформационныхТехнологий. http://www.citforum.ru /database/osbd /contents.shtml

2 Кузнецов С.Д. Основы баз данных. — 1-е изд. — М.: «Интернет-университет информационных технологий - ИНТУИТ.ру», 2005.

3 Дейт К.Дж. Введение в системы баз данных. — М.: Издательский дом «Вильямс», 2008.

4 Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. - Базы данных. Учебник для вузов. – М.: Корона-Принт, 2004.

5 Джоунс Э., Стивенз Р., Плю Р., Гарретт Р., Кригель А. Функции SQL. Справочник программиста. – М.: Диалектика, 2006.

Харрингтон Дж. Разработка баз данных. – М.: ДМК Пресс

6 Абдуллина В.З. Системы баз данных. Учебник. – Алматы: КазНТУ, 2009

7 Кумскова И. Базы данных.-М., 2012.

8 Советов Б.Я. Базы данных. Теория и практика.-М.: «Юрайт», 2012.

9 Смирнов С.Н. Практикум по работе с базами данных.-М.: Гелиос АРВ, 2012.

10 Виейра Р. Программирование баз данных Microsoft SQL Server 2005. Базовый курс.-М., 2007.

11 Проектирование баз данных. Конспект лекций. Ибраева Л.К.-Алматы: АУЭС, 2010.

12 Леонард Л. И др. Разработка приложений на основе Microsoft SQL Server 2008.-М., 2010