Учреждение образования
«Гомельский государственный университет имени Франциска Скорины»

Факультет физики и информационных технологий
Кафедра автоматизированных систем обработки информации

СОГЛАСОВАНО
Заведующий кафедрой
автоматизированных систем
обработки информации
А.В.Воруев
15 11 2022 г.

СОГЛАСОВАНО
Декан факультета физики
и информационных технологий
Д.Л.Коваленко
15 11 2022 г.

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

БАЗЫ И БАНКИ ДАННЫХ

для специальности
1-53 01 02 Автоматизированные системы обработки информации

Составители: Леванцов В.Н. – старший преподаватель кафедры автоматизированных систем обработки информации УО «Гомельский государственный университет им. Ф. Скорины»;
Пугачёва Е.Е. – старший преподаватель кафедры автоматизированных систем обработки информации УО «Гомельский государственный университет им. Ф. Скорины»;
Рафалова Е.В. – ассистент кафедры автоматизированных систем обработки информации УО «Гомельский государственный университет им. Ф. Скорины»

Рассмотрено и утверждено
на заседании кафедры АСОИ
15 ноября 2022 г., протокол № 4

Рассмотрено и утверждено
на заседании научно-методического
совета университета
01.12. 2022 г., протокол № 4

Гомель 2022

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Электронный учебно-методический комплекс (ЭУМК) по дисциплине компонента учреждения высшего образования «Базы и банки данных» представляет собой комплекс систематизированных учебных, методических и вспомогательных материалов, предназначенных для использования в образовательном процессе специальности 1-53 01 02 – Автоматизированные системы обработки информации.

ЭУМК разработан в соответствии со следующими нормативными документами:

1. Положением об учебно-методическом комплексе на уровне высшего образования, утвержденном постановлением Министерства образования Республики Беларусь от 26.07.2011 №167.
2. Образовательным стандартом (ОСВО 1-53 01 02 – 2021) Высшее образование. Первая ступень. Специальность 1-53 01 02 Автоматизированные системы обработки информации.
3. Учебной программой по учебной дисциплине «Базы и банки данных» для специальности 1-53 01 02 Автоматизированные системы обработки информации, утвержденной 17.05.2022, регистрационный номер № УД-2022-185/уч.

Цель создания ЭУМК – является подготовка студентов по основам проектирования и использования баз данных.

ЭУМК направлен на оказание помощи студентам в овладении теоретическими основами и практическими навыками проектирования и эксплуатации баз данных. Организация изучения дисциплины специализации на основе ЭУМК предполагает продуктивную образовательную деятельность, позволяющую сформировать социально-личностные и профессиональные компетенции будущих специалистов, обеспечить развитие познавательных и созидательных способностей личности.

ЭУМК способствует успешному осуществлению учебной деятельности, дает возможность планировать и осуществлять самостоятельную работу обучающихся, обеспечивает рациональное распределение учебного времени по темам учебной дисциплины и совершенствование методики проведения занятий.

В структурном отношении ЭУМК включает четыре раздела: теоретический, лабораторный практикум, раздел контроля знаний и вспомогательный.

Теоретический раздел содержит лекционный материал, включающий в себя 33 темы:

Вводная лекция

Раздел 1 Основные понятия и технологии работы с реляционными базами данных

Тема 1 Модели данных

Тема 2 Системы управления базами данных

Тема 3 Реляционная модель данных

Раздел 2 Работа с современными промышленными СУБД

Тема 1 Методы поиска и анализа информации в базе данных

Тема 2 СУБД MS Access. Работа в интерактивном режиме

Тема 3 Средства диалогового построения запросов

Раздел 3 Функциональные зависимости и поиск данных
Тема 1 Проблема аномалии и задача нормализации данных
Тема 2 Использование языка запросов SQL для выборки данных
Тема 3 Выборка данных из множества таблиц
Тема 4 Запросы обновления таблиц в языке запросов SQL

Раздел 4 Проектирование баз данных
Тема 1 Элементы проектирования баз данных
Тема 2 Проектирование баз данных
Тема 3 Описание структуры базы данных

Раздел 5 Работа с базами данных из современных языков программирования
Тема 1 Язык описания данных
Тема 2 Ссылочная целостность данных
Тема 3 Оптимизация работы с базами данных
Тема 4 Представления

Раздел 6 Работа с сетевыми базами данных
Тема 1 Особенности языка SQL для различных SQL-серверов
Тема 2 Создание таблиц баз данных в среде SQL-сервера
Тема 3 Установка связи с таблицами баз данных
Тема 4 Создание и использование представлений в среде SQL-сервера
Тема 5 Хранимые процедуры
Тема 6 Создание и использование курсоров
Тема 7 Триггеры SQL-сервера
Тема 8 Использование триггеров SQL-сервера
Тема 9 Транзакции
Тема 10 Основные понятия MySQL
Тема 11 Администрирование сетевых БД
Тема 12 Защита сетевых БД
Тема 13 Совместное использование данных и базы данных

Раздел 7 Базы данных в Интернет
Тема 1 Сервер Web как ядро приложений для Интернет
Тема 2 Связь приложений с базами данных через ODBC

Лабораторный раздел включает в себя темы лабораторных занятий. В каждой теме содержится краткое изложение теоретического материала и задания для выполнения по вариантам.

Вспомогательный раздел содержит необходимые элементы учебно-программной документации.

Все разделы ЭУМК в полной мере соответствуют содержанию и объему образовательного стандарта.

Дневная форма обучения (2 и 3 курс): Общее количество часов – 320; аудиторное количество часов – 140, из них 4 семестр: лекции – 26, лабораторных

занятий – 18 часов, управляемая самостоятельная работа – 8. Форма отчётности – зачет.

5 семестр: лекции – 20, лабораторных занятий – 58 часов, управляемая самостоятельная работа – 10. Курсовое проектирование – 30 часов. Форма отчётности – экзамен.

Заочная форма обучения (2 и 3 курс): Общее количество часов – 320, аудиторное количество часов – 36, из них 4 семестр: лекционных занятий – 10, лабораторных занятий – 8. Форма отчётности – зачет.

5 семестр: лекционных занятий – 10, лабораторных занятий – 8, курсовое проектирование – 30 часов.

Заочная форма обучения дистанционная (интегрированная на основе среднего специального образования) 3 и 4 курс: количество часов – 320 аудиторное количество часов – 20, из них 5 семестр: лекционных занятий – 2,

6 семестр: лекционных занятий – 4, лабораторных занятий – 6. Форма отчётности – зачет.

7 семестр лекционных занятий – 4, лабораторных занятий – 6, курсовое проектирование – 30 часов. Форма отчётности – экзамен.

ТЕКСТ ЛЕКЦИЙ

Вводная лекция

Раздел 1 Основные понятия и технологии работы с реляционными базами данных

Тема 1 Модели данных
Тема 2 Системы управления базами данных
Тема 3 Реляционная модель данных

Раздел 2 Работа с современными промышленными СУБД
Тема 1 Методы поиска и анализа информации в базе данных
Тема 2 СУБД MS Access. Работа в интерактивном режиме
Тема 3 Средства диалогового построения запросов

Раздел 3 Функциональные зависимости и поиск данных
Тема 1 Проблема аномалии и задача нормализации данных
Тема 2 Использование языка запросов SQL для выборки данных
Тема 3 Выборка данных из множества таблиц
Тема 4 Запросы обновления таблиц в языке запросов SQL

Раздел 4 Проектирование баз данных
Тема 1 Элементы проектирования баз данных
Тема 2 Проектирование баз данных
Тема 3 Описание структуры базы данных

Раздел 5 Работа с базами данных из современных языков программирования
Тема 1 Язык описания данных
Тема 2 Ссылочная целостность данных
Тема 3 Оптимизация работы с базами данных
Тема 4 Представления

Раздел 6 Работа с сетевыми базами данных
Тема 1 Особенности языка SQL для различных SQL-серверов
Тема 2 Создание таблиц баз данных в среде SQL-сервера
Тема 3 Установка связи с таблицами баз данных
Тема 4 Создание и использование представлений в среде SQL-сервера
Тема 5 Хранимые процедуры
Тема 6 Создание и использование курсоров
Тема 7 Триггеры SQL-сервера
Тема 8 Использование триггеров SQL-сервера
Тема 9 Транзакции
Тема 10 Основные понятия MySQL
Тема 11 Администрирование сетевых БД
Тема 12 Защита сетевых БД
Тема 13 Совместное использование данных и базы данных

Раздел 7 Базы данных в Интернет
Тема 1 Сервер Web как ядро приложений для Интернет
Тема 2 Связь приложений с базами данных через ODBC

ВВОДНАЯ ЛЕКЦИЯ

Информация и данные. Эволюция концепции обработки и хранения данных.
Базы и банки данных. Система управления базами данных (СУБД).

Восприятие реального мира можно соотнести с последовательностью разных, хотя иногда и взаимосвязанных, явлений. С давних времен люди пытались описать эти явления (даже тогда, когда не могли их понять). Такое описание называют **данными**.

Традиционно фиксация данных осуществляется с помощью конкретного средства общения (например, с помощью естественного языка или изображений) на конкретном носителе (например, камне или бумаге). Обычно данные (факты, явления, события, идеи или предметы) и их интерпретация (семантика) фиксируются совместно, так как естественный язык достаточно гибок для представления того и другого. Примером может служить утверждение "Стоимость авиабилета 128". Здесь "128" – данное, а "Стоимость авиабилета" – его семантика.

Нередко данные и интерпретация разделены. Например, "Расписание движения самолетов" может быть представлено в виде таблицы, в верхней части которой (отдельно от данных) приводится их интерпретация (шапка). Такое разделение затрудняет работу с данными.

Применение ЭВМ для ведения *(Ведение (сопровождение, поддержка) данных – термин объединяющий действия по добавлению, удалению или изменению хранимых данных)* и обработки данных обычно приводит к еще большему разделению данных и интерпретации. ЭВМ имеет дело главным образом с данными как таковыми. Большая часть интерпретирующей информации вообще не фиксируется в явной форме.

Существует по крайней мере две исторические причины, по которым применение ЭВМ привело к отделению данных от интерпретации.

Во-первых, ЭВМ не обладали достаточными возможностями для обработки текстов на естественном языке – основном языке интерпретации данных.

Во-вторых, стоимость памяти ЭВМ была первоначально весьма велика. Память использовалась для хранения самих данных, а интерпретация традиционно возлагалась на пользователя. Пользователь закладывал интерпретацию данных в свою программу. Это существенно повышало роль программы, так как вне интерпретации данные представляют собой не более чем совокупность битов на запоминающем устройстве.

Жесткая зависимость между данными и использующими их программами создает серьезные проблемы в ведении данных и делает использования их менее гибкими.

С самого начала развития вычислительной техники образовались два основных направления ее использования. Первое направление - применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную.

Второе направление - использование средств вычислительной техники в автоматических или автоматизированных информационных системах.

В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. (банковские системы, системы резервирования билетов и т.д).

На самом деле, второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными возможностями в части памяти. Понятно, что **говорить о надежном и долговременном хранении информации можно только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания.**

В начале использовались два вида устройств внешней памяти: магнитные ленты и барабаны. При этом емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали **последовательный доступ к данным**. Магнитные же барабаны обеспечивают **произвольный доступ к данными** (но были ограниченного размера).

Для информационных систем, в которых потребность в текущих данных определяется пользователем, наличие только магнитных лент и барабанов неудовлетворительно. Одним из естественных требований к таким системам является средняя быстрота выполнения операций.

С появлением магнитных дисков началась история систем управления данными во внешней памяти.

До этого **каждая прикладная программа**, которой требовалось хранить данные во внешней памяти, **сама определяла расположение каждой порции данных** на магнитной ленте или барабане и выполняла обмены между оперативной и внешней памятью с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ операционной системы). Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти.

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

Файловые системы обычно обеспечивают хранение слабо структурированной информации, оставляя дальнейшую структуризацию прикладным программам. Это позволяет при разработке любой новой прикладной системы, опираясь на простые, стандартные и сравнительно дешевые средства файловой системы, реализовать те структуры хранения, которые наиболее естественно соответствуют специфике данной прикладной области.

На начальном этапе использования вычислительной техники для управления информацией **проблемы структуризации данных решались индивидуально в каждой информационной системе**. Производились необходимые надстройки над файловыми системами (библиотеки программ), подобно тому, как это делается в компиляторах, редакторах и т.д.

Но поскольку информационные системы требуют сложных структур данных, эти дополнительные индивидуальные средства управления данными являлись существенной частью информационных систем и практически повторялись от одной системы к другой. Очень скоро стало понятно, что невозможно обойтись общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения данных.

Существует несколько потребностей, которые не покрываются возможностями систем управления файлами: поддержание логически согласованного набора файлов; обеспечение языка манипулирования данными; восстановление информации после разного рода сбоев; реально параллельная работа нескольких пользователей.

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 - базы данных (БД) и банки данных.

База данных— совокупность экземпляров различных типов записей и отношений между записями, агрегатами данных, элементами данных.

Система баз данных - В большинстве систем термин *база данных* относится не ко *всем* типам записей, а только к некоторой их совокупности. В одной системе может использоваться несколько баз данных, однако предполагается, что различные базы данных разделены и не связаны между собой. Для обозначения совокупности баз данных требуется некоторый термин — мы будем использовать в этих случаях термин *система баз данных.*

Иногда для обозначения совокупности баз данных употребляется термин *банк данных.* Некоторые специалисты вместо *базы данных* используют термин *банк данных,* а под базой данных подразумевают совокупность банков данных. Часто в литературе термин *банк данных* используется без четкого определения.

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

Логически в современной реляционной СУБД можно выделить наиболее внутреннюю часть - ядро СУБД (часто его называют Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения, набор утилит. В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL (или в подсистеме поддержки выполнения таких программ) и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. При использовании архитектуры "клиент-сервер" ядро является основной составляющей серверной части системы.

Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу.

Наконец, в отдельные утилиты БД обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

Модель данных. Объект. Атрибут. Тип и экземпляр. Схема и подсхема.
Уровни абстрагирования процессов обработки данных.

Модель данных описывает некоторый набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, если они основываются на этой модели. Наличие модели данных позволяет сравнивать конкретные реализации, используя один общий язык.

Объекты и атрибуты

Элементы, информацию о которых мы сохраняем, будем называть *объектами.* Объект может быть материальным (например, служащий, изделие или населенный пункт) и нематериальным (например, событие, название задания, счет покупателя, индекс прибыли или абстрактная идея). Объект имеет различные свойства, которые можно запоминать (например, цвет, денежная величина или имя). При обработке данных часто имеют дело с совокупностью однородных объектов (таких, как служащие) и записывают информацию об одних и тех же свойствах каждого из них. Мы будем называть совокупность однородных объектов *набором объектов.*

Определения, данные выше, относятся к реальному миру.

Три области

Выделяются три области, о которых можно говорить при обсуждении понятия *информация*. Первая область — реальный мир, в котором объекты существуют и имеют определенные свойства. Вторая область — область идей и информации, существующих в представлении людей и программистов. Здесь говорят об атрибутах объектов и обозначают атрибуты символически: на естественном языке или языке программирования; атрибутам приписывают значения. Третья область представляет собой область, в которой используются строки символов или битов для кодирования элементов информации. Эти три области представлены на рис. 5.2. Третью область затем можно разделить на данные в представлении прикладного программиста, общую логическую структуру данных в представлении администратора данных и на физическое представление данных.

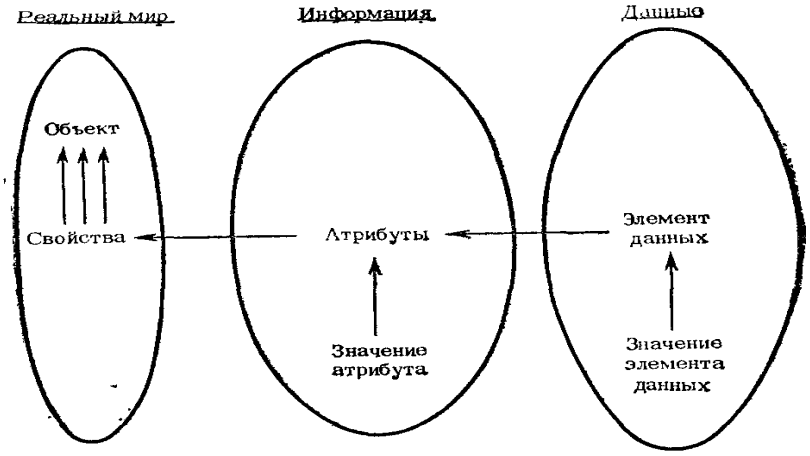


Рис. 5.2. Три области, о которых говорится при обсуждении понятия информации.

Совокупность битов или символов, представляющая значение конкретного элемента данных, должна быть связана с тем элементом данных, который имеет это значение. Элемент данных представляет атрибут, и атрибут должен быть связан с соответствующим объектом. Один атрибут имеет конкретное значение, которым он *идентифицирует* объект. Совокупность битов или символов, представляющая значение одного элемента данных, может существовать независимо от информации, которая запоминается с их помощью. Она имеет смысл только тогда, когда связана с элементами данных, представленными этими значениями. Мы можем, например, постоянно хранить значения элемента данных: голубой, оранжевый и фиолетовый с зелеными полосами; эти значения могут быть впоследствии связаны с некоторыми конкретными элементами данных.

Модели данных: схемы и подсхемы

Если бы назначением базы данных было только хранение данных, то структура ее была бы простой. Причина же ее сложности объясняется тем, что она должна обеспечивать еще и связи между различными элементами данных.

Минимальным фрагментом данных является *элемент данных* (поле, элемент). Элемент данных не может подразделяться на меньшие типы данных, не теряя при этом смысла для пользователя; это атом данных (и состоит он из таких частиц, как биты и байты). В последующих диаграммах для" представления элемента данных будем использовать эллипс, внутри которого будем записывать имя типа элемента данных. Например, (рис.1А)

рис. 1 А

рис. 1 Б

Сам по себе элемент данных ничего не представляет. Он приобретает смысл только тогда, когда он связан с другими элементами данных. Эта связь изображается следующим образом (рис. 1 Б)

База данных состоит из элементов данных и связей между ними. В базе данных много различных типов элементов данных, и поэтому необходима специальная схема, позволяющая изобразить связи между типами элементов данных. Эта схема иногда называется *моделью данных.* Существуют различные способы изображения такой схемы. На диаграмме, которую мы используем здесь, не показан способ *физического* хранения данных. На ней показаны только *логические* связи между элементами данных. Официальная схема Лондонского метрополитена не имеет отношения к физическому расположению путей и станций. На ней не показаны реальные изгибы путей и реальные расстояния между станциями. Подобно схеме базы данных, на ней просто представлены связи между станциями. Ее можно рассматривать как *модель* реального мира, которая не имеет в общем-то большого сходства с действительностью, но которая может быть полезной ее пользователям в качестве одного из представлений реального мира.

Описанию *физического* размещения данных должно предшествовать рассмотрение моделей данных, необходимых пользователям. Эти модели являются *логическим* представлением данных. Лондонский метрополитен можно представить в виде *логического* описания транспортной

системы. Строители могут изменить физические пути, прокладывая дорогу над Темзой, а не под ней, но при этом логическая схема не изменится.

Прежде чем описывать *физическую* реализацию связей между данными, мы должны обсудить способ, с помощью которого конечные пользователи базы данных (обращающиеся к ней с помощью терминалов или прикладных программ) представляют эти связи. Существуют различные способы представления *логических* связей. Одни из них вполне удовлетворительны, другие довольно запутанны и могут ввести в заблуждение, третьи ограниченны в том смысле, что с их помощью нельзя представить все реально существующие связи. Некоторые способы негибки, т. е. не позволяют легко представить расширения и изменения данных в случае возможного развития базы данных. В этой и в последующих главах обсуждается *логическое* представление данных.

Способность программных средств управления данными отделить физическую структуру данных от представления пользователей или от «логической» организации данных дает пользователям возможность (по крайней мере, теоретически) представлять логическую структуру данных *независимо от их физической реализации.* Представление пользователей о структуре данных должно быть описано в любой удобной для него и его коллег (настоящих и будущих) форме; средства управления данными должны осуществлять преобразование логической структуры в любую физическую структуру данных, обеспечивающую высокую производительность.

В практике разработки и использования вычислительных машин еще не существует идеальных программных средств управления данными, и поэтому, как мы увидим ниже, выбирают компромиссное решение: раздельное описание логической и физической структур.

Схема

Структуру данных необходимо описывать формальным образом. Описания логической и физической структур базы данных используются программными средствами управления базами данных при обработке требований пользователей на получение той информации, которую содержит база данных. Описание общей логической структуры базы данных называют *схемой [*Термины *схема* и *подсхема* были введены рабочей группой по базам данных CODASYL. Они, однако, оказались подходящими и для описания данных в системах баз данных, не отвечающих требованиям CODASYL. Эти термины получили широкое распространение*]*. Ее называют иногда *общей моделью* данных, *концептуальной моделью* или *концептуальной схемой.* Эти термины примерно равнозначны. Схема представляет собой таблицу типов используемых данных. Она содержит имена объектов и их атрибуты и определяет существующую между ними связь. Схема представляет собой структуру, в которой могут быть помещены значения элементов данных. Подобно табло в аэропорту, на котором высвечивается информация о прибытии и отправлении самолетов, схема не меняется, в то время как величины, помещенные в ней, время от времени изменяются.

Мы должны различать понятия *тип записи* и *экземпляр записи,* Запись (подобно схеме) — такая структура, в которую можно помещать конкретные значения данных. Запись с конкретными значениями данных в текущий момент времени может быть названа *экземпляром записи.*

Подобное отличие между записью и экземпляром записи существует и для элементов данных, и для агрегатов данных, и для всех других категории данных.

Изображения схем и записей можно представлять в виде расположенных одного за другим прямоугольников так, чтобы они определяли все значения каждого элемента данных.

Подсхемы

Термин *схема* используется для определения полной таблицы всех типов элементов данных и типов записей, хранимых в базе данных. Термином *подсхема* определяют описание данных, которое использует прикладной программист. На основе одной схемы можно составить много различных подсхем.

Прикладной программист или конечный пользователь необязательно должен знать о схеме базы данных в целом, так как обычно она очень сложна. Иногда такая неосведомленность объясняется соображениями безопасности. Программист или пользователь должен иметь дело только с теми конкретными приложениями и записями, которые ему нужны.

Администратор базы данных должен гарантировать, что используемые прикладными программистами подсхемы могут быть получены из схемы. Программы управления базой данных автоматически получают данные, соответствующие подсхеме, на основе данных, описанных в схеме, и передают их прикладной программе.

Три типа описания данных

Ни схемы, ни подсхемы не отражают способов физического хранения данных. Для заданной логической структуры возможны различные формы физической организации данных.

Итак, существуют три различных вида описания данных:

1. **Подсхема**—таблица, описывающая ту часть данных, которая ориентирована на нужды одной или нескольких прикладных программ (организация файлов программиста).

2. **Глобальное описание логической структуры базы данных**, или **схема**,— таблица, логически описывающая всю базу данных. Она отражает представление о данных администратора данных или тех системных аналитиков, которые работают со всей базой данных.

3. **Описание физической организации базы данных** — таблица физического расположения данных на носителях информации. Это представление о данных нужно системному программисту или системному разработчику, которые занимаются вопросами эффективности работы системы, расположения данных на носителях, их индексирования или поиска, а также вопросами использования методов сжатия данных.

Подсхему иногда называют *частным представлением* или LVIEW. Одна подсхема может обслуживать несколько прикладных программ и может быть определена отдельно от программ, так как это сделано, например, в системе IMS фирмы IBM, где она размещается в блоке PSB — блоке спецификации программы. Для определения подсхемы используется также термин *подмодель.* Исследовательская группа Американского национального института стандартов по системам управления базами данных ANSI/X3/SPARC опубликовала отчет в 1975 г., в котором определила три уровня описания данных: *внешнюю схему, концептуальную схему* и *внутреннюю схему.* В настоящее время эти термины широко используются.

Часто используют также и четвертый вид описания данных — для конечного пользователя. Описание данных, которое система передает пользователю терминала, должно быть как можно более близким к тому описанию данных, которое он использует в своей работе. Способ, с помощью которого описание передается пользователю терминала, зависит от возможностей, запроектированных для диалога человек — машина.

Отметим, что система управления базами данных не использует сразу все существующие в системе описания данных, прежде всего в интересах самих описаний. Она допускает в случае необходимости изменение одного описания, в то время как другие описания *сохраняются неизменными.* Только такой принцип позволяет избежать необходимости изменять одновременно все описания данных.

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**.

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 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.

Иерархические модели данных. Сетевые модели данных. Объектно-ориентированные модели данных

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.

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 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.

Некоторые наиболее общие характеристики ранних систем:

1. Эти системы активно использовались в течение многих лет, дольше, чем используется какая-либо из реляционных СУБД. На самом деле некоторые из ранних систем используются даже в наше время, накоплены громадные базы данных, и одной из актуальных проблем информационных систем является использование этих систем совместно с современными системами.
2. Все ранние системы не основывались на каких-либо абстрактных моделях. Понятие модели данных фактически вошло в обиход специалистов в области БД только вместе с реляционным подходом. Абстрактные представления ранних систем появились позже на основе анализа и выявления общих признаков у различных конкретных систем.

3. В ранних системах доступ к БД производился на уровне записей. Пользователи этих систем осуществляли явную навигацию в БД, используя языки программирования, расширенные функциями СУБД. Интерактивный доступ к БД поддерживался только путем создания соответствующих прикладных программ с собственным интерфейсом.
4. Можно считать, что уровень средств ранних СУБД соотносится с уровнем файловых систем примерно так же, как уровень языка Кобол соотносится с уровнем языка Ассемблера. Заметим, что при таком взгляде уровень реляционных систем соответствует уровню языков Ада или APL.
5. Навигационная природа ранних систем и доступ к данным на уровне записей заставляли пользователя самого производить всю оптимизацию доступа к БД, без какой-либо поддержки системы.
6. После появления реляционных систем большинство ранних систем было оснащено "реляционными" интерфейсами. Однако в большинстве случаев это не сделало их по-настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

Иерархические системы

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.

Между типами записи поддерживаются связи.

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:

Найти указанное дерево БД (например, отдел 310);

Перейти от одного дерева к другому;

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

Перейти от одной записи к другой в порядке обхода иерархии;

Вставить новую запись в указанную позицию;

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.

Сетевые системы

Типичным представителем является Integrated Database Management System (IDMS) компании Cullinet Software, Inc., предназначенная для использования на машинах основного класса фирмы IBM под управлением большинства операционных систем. Архитектура системы основана на предложениях Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL), организации, ответственной за определение языка

программирования Кобол. Отчет DBTG был опубликован в 1971 г., а в 70-х годах появилось несколько систем, среди которых IDMS.

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:

- Каждый экземпляр типа P является предком только в одном экземпляре L;
- Каждый экземпляр C является потомком не более, чем в одном экземпляре L.

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

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

- Данный тип записи P может быть типом записи предка в любом числе типов связи.
- Данный тип записи P может быть типом записи потомка в любом числе типов связи.

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.

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

Предок и потомок могут быть одного типа записи.

Манипулирование данными

Примерный набор операций может быть следующим:

1. Найти конкретную запись в наборе однотипных записей (инженера Сидорова);
2. Перейти от предка к первому потомку по некоторой связи (к первому сотруднику отдела 310);
3. Перейти к следующему потомку в некоторой связи (от Сидорова к Иванову);
4. Перейти от потомка к предку по некоторой связи (найти отдел Сидорова);
5. Создать новую запись;
6. Уничтожить запись;
7. Модифицировать запись;
8. Включить в связь;
9. Исключить из связи;
10. Переставить в другую связь и т.д.

Ограничения целостности

В принципе их поддержание не требуется, но иногда требуют целостности по ссылкам (как в иерархической модели).

Достоинства и недостатки

Сильные места ранних СУБД:

- Развитые средства управления данными во внешней памяти на низком уровне;
- Возможность построения вручную эффективных прикладных систем;
- Возможность экономии памяти за счет разделения подобъектов (в сетевых системах).

Недостатки:

- Слишком сложно пользоваться;
- Фактически необходимы знания о физической организации;
- Прикладные системы зависят от этой организации;
- Их логика перегружена деталями организации доступа к БД.

***** Объектно-ориентированные БД

В объектно-ориентированной модели отдельные записи базы данных представляются в виде объектов. Между записями БД и функциями их обработки устанавливаются взаимосвязи с помощью механизмов, подобных соответствующим средствам в объектно-ориентированных языках программирования. Объектно-ориентированные модели сочетают особенности сетевой и реляционной моделей и используются для создания крупных БД со сложными структурами данных.

Объектно-ориентированное программирование в СУБД.

В наиболее общей и классической постановке объектно-ориентированный подход базируется на концепциях:

- объекта и идентификатора объекта;
- атрибутов и методов;
- классов;
- иерархии и наследования классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Каждый объект имеет состояние и поведение. Состояние объекта - набор значений его атрибутов. Поведение объекта - набор методов (программный код), оперирующих над состоянием объекта. Значение атрибута объекта - это тоже некоторый объект или множество объектов. Состояние и поведение объекта инкапсулированы в объекте; взаимодействие между объектами производится на основе передачи сообщений и выполнении соответствующих методов. Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Допускается порождение нового класса на основе уже существующего класса - наследование.

Объектно-ориентированное программирование после своего появления начало быстро завоевывать популярность среди программистов. Поэтому создатели СУБД также постарались включить элементы объектно-ориентированного подхода в свои языки программирования.

Например, еще в СУБД Clipper 5.0 существовали следующие классы объектов:

Класс Error - представляет объекты, содержащие информацию об ошибках выполнения программ.

Класс Get - общий механизм для редактирования данных. Get-объекты предоставляют усложненную архитектуру для форматирования и редактирования данных, в том числе управления курсором и проверки данных. Проверка данных выполняется через поставляемые пользователем блоки кода, форматирование может управляться с использованием стандартных строк шаблона.

Класс TBrowse - является аппаратом полноэкранного редактирования табличных данных. TBrowse-объекты обеспечивают развитые структуры для ввода, форматирования и представления данных. Поиск данных и просмотр файлов осуществляются определенным пользователем кодовыми блоками, что обеспечивает большую гибкость и взаимодействие между механизмом полноэкранного редактирования и данными. Формат единичных элементов данных может полностью определяться блоками кодов поиска данных. Все атрибуты и форматы представления данных на экране могут задаваться посылкой соответствующих сообщений TBrowse-объектам.

Основу TBrowse-объектов составляют один или несколько TBcolumn объектов. TBcolumn-объекты содержат информацию, необходимую для определения отдельного столбца полноэкранной таблицы

TBColumn-объект содержит всю необходимую информацию для того, чтобы определить один столбец TBrowse-объекта.

Как видно, все эти объекты касались в бОльшей степени интерфейса. Создавать свои классы в Clipper программист не мог.

Более широкий набор объектов для разработки экранных форм и отчетов предоставляют библиотеки таких СУБД как Visual FoxPro, MS Access и др. – большинство современных СУБД.

С одной стороны, это упрощает и ускоряет разработку интерфейса приложения для работы с БД, а с другой стороны, содержит одну большую проблему: в то время как само приложение разрабатывается на основании объектно-ориентированного подхода, сама база данных основана на реляционном (иерархическом, сетевом) подходе. Таким образом, логически связанный проект распадается на две реализационно различные части: проектирование и разработка схемы базы данных и проектирование и разработка программного кода приложения. Понятно, что, как правило, данные, хранящиеся во внешней памяти, никому не нужны без обрабатывающих их программ, а эти программы бессмысленны при отсутствии данных. Тем не менее, при реализации этих связанных частей используются абсолютно разные модели и инструменты: системы программирования, с одной стороны, и модели и языки баз данных - с другой.

В качестве решения этого противоречия возникла идея создания ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ БД.

Направление объектно-ориентированных баз данных (ООБД) возникло сравнительно давно. Публикации появлялись уже в середине 1980-х. Однако наиболее активно это направление развивается в последние годы. С каждым годом увеличивается число публикаций и реализованных коммерческих и экспериментальных систем. основная цель проектировщиков и разработчиков ОО СУБД состояла в том, чтобы предоставить разработчикам информационных приложений механизм управления данными во внешней памяти, который полностью стыковался бы со средствами объектно-ориентированного программирования.

Технология ООСУБД предполагает существование интегрированной языковой среды, которая одновременно позволяет конструировать объектную базу данных, содержащую не только данные, но и программный код (методы объектов), обеспечивающий доступ к этим данным, и код приложения. Тем самым, исчезает разрыв между пассивными данными и активными программами, проект прикладной системы ведется в рамках единой технологии, что убыстряет его разработку и облегчает последующее сопровождение. Естественно, что при этом должны преследоваться цели сохранения всех преимуществ объектно-ориентированного программирования (уникальная идентификация объектов, инкапсуляция, наследование, полиморфизм и т.д.) и систем баз данных (многопользовательский режим доступа, восстановление после сбоев, управление транзакциями и т.д.).

Основные трудности объектно-ориентированного моделирования данных проистекают из того, что такого развитого математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных, не существует. В большой степени поэтому до сих пор нет базовой объектно-ориентированной модели.

Тем не менее, проблемой объектно-ориентированных БД занимаются достаточно давно, и уже есть много примеров удачного использования этих продуктов в реализованных приложениях. На основании этих практических наработок стало возможным выделить основные принципы, на которых могут основываться ОО СУБД:

1) предлагается выделить два уровня моделирования объектов: нижний (структурный) и верхний (поведенческий). На структурном уровне поддерживаются сложные объекты, их идентификация и разновидности связи. База данных - это набор элементов данных, связанных отношениями "входит в класс" или "является атрибутом". Таким образом, БД может рассматриваться как ориентированный граф.

2) важным аспектом является четкое разделение схемы БД и самой БД. В качестве первичных концепций схемного уровня ООБД выступают типы и классы.

3) для точного определения ООБД требуется уровень мета-схемы, содержимое которой должно определять виды объектов и связей, допустимых на схемном уровне БД. Мета-схема должна играть для ООБД такую же роль, какую играет структурная часть реляционной модели данных для схем реляционных баз данных.

Еще одной проблемой при разработке ОО СУБД является выбор внутреннего языка программирования для такой системы.

Примерами успешной реализации ОО СУБД могут служить:

O2 – производитель Ardent Software, Inc. (в ней реализован свой собственный функциональный объектно-ориентированный язык программирования, а также используются два объектно-ориентированных расширения языков Бейсик и Си, наибольшее распространение из которых получил основанный на C++ язык CO2 - не является полностью самостоятельным языком и предназначен для программирования методов ранее определенных классов.)

GemStone/S – разрабочик GemStone Systems, Inc (основанный на языке SmallTalk)

проект ORION (наиболее интересный проект, базирующийся на языке Лисп (Common Lisp), являющемся в этой СУБД и инструментальным языком, и базой объектно-ориентированного языка программирования)

система VBASE (наряду со специально разработанным языком TDL, предназначенным для определения типов, используется объектно-ориентированное расширение языка Си - COP (C Object Processor))

кроме того можно упомянуть СУБД Cashe (InterSystems Corporation), ObjectStore (Object Design, Inc), Objectivity/DB (Objectivity, Inc), POET (POET Software, Inc.), VERSANT (Versant Corporation) и т.д.

Однако следует отметить, что крупные софтверные компании, такие как Oracle, Informix, Sybase, Microsoft и IBM, не собираются развивать свою линию продуктов ООСУБД. Вместо этого они предлагают свои подходы к расширению реляционных баз данных объектными свойствами.

Реляционные модели данных. Отношения. Ключи.
Реляционное исчисление. Реляционная алгебра.

Реляционные базы данных.

Этот подход является наиболее распространенным в настоящее время, хотя наряду с общепризнанными достоинствами обладает и рядом недостатков. К числу достоинств реляционного подхода можно отнести:

1. наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;

2. наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации баз данных;
3. возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

Реляционные системы далеко не сразу получили широкое распространение. В то время, как основные теоретические результаты в этой области были получены еще в 70-х, и тогда же появились первые прототипы реляционных СУБД, долгое время считалось невозможным добиться эффективной реализации таких систем. Однако отмеченные выше преимущества и постепенное накопление методов и алгоритмов организации реляционных баз данных и управления ими привели к тому, что уже в середине 80-х годов реляционные системы практически вытеснили с мирового рынка ранние СУБД.

В настоящее время основным предметом критики реляционных СУБД является не их недостаточная эффективность, а присущая этим системам некоторая ограниченность (прямое следствие простоты) при использование в так называемых нетрадиционных областях (наиболее распространенными примерами являются системы автоматизации проектирования), в которых требуются предельно сложные структуры данных. Еще одним часто отмечаемым недостатком реляционных баз данных является невозможность адекватного отражения семантики предметной области. Другими словами, возможности представления знаний о семантической специфике предметной области в реляционных системах очень ограничены. Современные исследования в области постреляционных систем главным образом посвящены именно устранению этих недостатков.

Базовые понятия реляционных баз данных

Основными понятиями реляционных баз данных являются тип данных, домен, атрибут, кортеж, первичный ключ и отношение.

Тип данных

Понятие тип данных в реляционной модели данных полностью адекватно понятию типа данных в языках программирования. Обычно в современных реляционных БД допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как "деньги"), а также специальных "темпоральных" данных (дата, время, временной интервал). Достаточно активно развивается подход к расширению возможностей реляционных систем абстрактными типами данных (соответствующими возможностями обладают, например, системы семейства Ingres/Postgres

Домен

Понятие домена более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат "истина", то элемент данных является элементом домена.

Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа.

// Например, домен "Имена" в нашем примере определен на базовом типе строк символов, но в число его значений могут входить только те строки, которые могут изображать имя (в частности, такие строки не могут начинаться с мягкого знака).

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену.

// В нашем примере значения доменов "Номера пропусков" и "Номера групп" относятся к типу целых чисел, но не являются сравнимыми. Заметим, что в большинстве реляционных СУБД понятие домена не используется, хотя в Oracle V.7 оно уже поддерживается.

Схема отношения, схема базы данных

Схема отношения - это именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}. Степень или "арность" схемы отношения - мощность этого множества. Если все атрибуты одного отношения определены на разных доменах, осмысленно использовать для именования атрибутов имена соответствующих доменов (не забывая, конечно, о том, что это является всего лишь удобным способом именования и не устраняет различия между понятиями домена и атрибута).

Схема БД (в структурном смысле) - это набор именованных схем отношений.

Кортеж, отношение

Кортеж, соответствующий данной схеме отношения, - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым, степень или "арность" кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения. Попросту говоря, кортеж - это набор именованных значений заданного типа.

Отношение - это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отношение-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения. На самом деле, понятие схемы отношения ближе всего к понятию структурного типа данных в языках программирования. Было бы вполне логично разрешать отдельно определять схему отношения, а затем одно или несколько отношений с данной схемой.

Однако в реляционных базах данных это не принято. Имя схемы отношения в таких базах данных всегда совпадает с именем соответствующего отношения-экземпляра. В классических реляционных базах данных после определения схемы базы данных изменяются только отношения-экземпляры. В них могут появляться новые и удаляться или модифицироваться существующие кортежи. Однако во многих реализациях допускается и изменение схемы базы данных: определение новых и изменение существующих схем отношения. Это принято называть **эволюцией схемы базы данных.**

Обычным житейским представлением отношения является таблица, заголовком которой является схема отношения, а строками - кортежи отношения-экземпляра; в этом случае имена атрибутов именуют столбцы этой таблицы. Поэтому иногда говорят "столбец таблицы", имея в виду "атрибут отношения". Когда мы перейдем к рассмотрению практических вопросов организации реляционных баз данных и средств управления, мы будем использовать эту житейскую терминологию. Этой терминологии придерживаются в большинстве коммерческих реляционных СУБД.

Реляционная база данных - это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

Как видно, основные структурные понятия реляционной модели данных (если не считать понятия домена) имеют очень простую интуитивную интерпретацию, хотя в теории реляционных БД все они определяются абсолютно формально и точно.

Фундаментальные свойства отношений

Остановимся теперь на некоторых важных свойствах отношений, которые следуют из приведенных ранее определений:

Отсутствие кортежей-дубликатов

То свойство, что отношения не содержат кортежей-дубликатов, следует из определения отношения как множества кортежей. В классической теории множеств по определению каждое множество состоит из различных элементов.

Из этого свойства вытекает наличие у каждого отношения так называемого первичного ключа - набора атрибутов, значения которых однозначно определяют кортеж отношения. Для каждого отношения по крайней мере полный набор его атрибутов обладает этим свойством. Однако при формальном определении первичного ключа требуется обеспечение его "минимальности", т.е. в набор атрибутов первичного ключа не должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства - однозначно определять кортеж. Понятие *первичного ключа* является исключительно важным в связи с понятием целостности баз данных.

Заметим, что во многих практических реализациях РСУБД допускается нарушение свойства уникальности кортежей для промежуточных отношений, порождаемых неявно при выполнении запросов. Такие отношения являются не множествами, а мультимножествами, что в ряде случаев позволяет добиться определенных преимуществ, но иногда приводит к серьезным проблемам.

Отсутствие упорядоченности кортежей

Свойство отсутствия упорядоченности кортежей отношения также является следствием определения отношения-экземпляра как множества кортежей. Отсутствие требования к поддержанию порядка на множестве кортежей отношения дает дополнительную гибкость СУБД при хранении баз данных во внешней памяти и при выполнении запросов к базе данных. Это не противоречит тому, что при формулировании запроса к БД, например, на языке SQL можно потребовать сортировки результирующей таблицы в соответствии со значениями некоторых

столбцов. Такой результат, вообще говоря, не отношение, а некоторый упорядоченный список кортежей.

Отсутствие упорядоченности атрибутов

Атрибуты отношений не упорядочены, поскольку по определению схема отношения есть множество пар {имя атрибута, имя домена}. Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута. Это свойство теоретически позволяет, например, модифицировать схемы существующих отношений не только путем добавления новых атрибутов, но и путем удаления существующих атрибутов. Однако в большинстве существующих систем такая возможность не допускается, и хотя упорядоченность набора атрибутов отношения явно не требуется, часто в качестве неявного порядка атрибутов используется их порядок в линейной форме определения схемы отношения.

Атомарность значений атрибутов

Значения всех атрибутов являются атомарными. Это следует из определения домена как потенциального множества значений простого типа данных, т.е. среди значений домена не могут содержаться множества значений (отношения). Принято говорить, что в реляционных базах данных допускаются только нормализованные отношения или отношения, представленные в первой нормальной форме.

Нормализованные отношения составляют основу классического реляционного подхода к организации баз данных. Они обладают некоторыми ограничениями (не любую информацию удобно представлять в виде плоских таблиц), но существенно упрощают манипулирование данными.

Реляционная модель данных

Определение реляционной модели по Дейту

Наиболее распространенная трактовка реляционной модели данных, по-видимому, принадлежит Дейту, который воспроизводит ее (с различными уточнениями) практически во всех своих книгах.

Согласно Дейту реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной части, манипуляционной части и целостной части.

В структурной части модели фиксируется, что единственной структурой данных, используемой в реляционных БД, является нормализованное n-арное отношение. По сути дела, в предыдущих двух разделах этой лекции мы рассматривали именно понятия и свойства структурной составляющей реляционной модели.

В манипуляционной части модели утверждаются два фундаментальных механизма манипулирования реляционными БД - реляционная алгебра и реляционное исчисление. Первый механизм базируется в основном на классической теории множеств (с некоторыми уточнениями), а второй - на классическом логическом аппарате исчисления предикатов первого порядка. Мы рассмотрим эти механизмы более подробно на следующей лекции, а пока лишь заметим, что основной функцией манипуляционной части реляционной модели является обеспечение меры реляционности любого конкретного языка реляционных БД: язык называется реляционным, если он обладает не меньшей выразительностью и мощностью, чем реляционная алгебра или реляционное исчисление.

Наконец, **в целостной части реляционной модели** данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД.

Первое требование называется **требованием целостности сущностей**.

Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е. другими словами, **любое отношение должно обладать первичным ключом.** Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Второе требование называется **требованием целостности по ссылкам** и является несколько более сложным. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений.

Например, представим, что нам требуется представить в реляционной базе данных сущность ОТДЕЛ с атрибутами ОТД_НОМЕР (номер отдела), ОТД_КОЛ (количество сотрудников) и ОТД_СОТР (набор сотрудников отдела). Для каждого сотрудника нужно хранить СОТР_НОМЕР (номер сотрудника), СОТР_ИМЯ (имя сотрудника) и СОТР_ЗАРП (заработная плата сотрудника). Как мы вскоре увидим, при правильном проектировании соответствующей БД в ней появятся два

отношения: ОТДЕЛЫ (ОТД_НОМЕР, ОТД_КОЛ) (первичный ключ - ОТД_НОМЕР) и СОТРУДНИКИ (СОТР_НОМЕР, СОТР_ИМЯ, СОТР_ЗАРП, СОТР_ОТД_НОМ) (первичный ключ - СОТР_НОМЕР).

Как видно, атрибут СОТР_ОТД_НОМ появляется в отношении СОТРУДНИКИ не потому, что номер отдела является собственным свойством сотрудника, а лишь для того, чтобы иметь возможность восстановить при необходимости полную сущность ОТДЕЛ. Значение атрибута СОТР_ОТД_НОМ в любом кортеже отношения СОТРУДНИКИ должно соответствовать значению атрибута ОТД_НОМ в некотором кортеже отношения ОТДЕЛЫ.

Атрибут такого рода называется **внешним ключом**, поскольку его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения (т.е. задают значения их первичного ключа).

Говорят, что отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором такой же атрибут является первичным ключом.

Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать). Для нашего примера это означает, что если для сотрудника указан номер отдела, то этот отдел должен существовать.

Ограничения целостности сущности и по ссылкам должны поддерживаться СУБД. **Для соблюдения целостности сущности достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа.** С целостностью по ссылкам дела обстоят несколько более сложно.

Понятно, что при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа. Но как быть при удалении кортежа из отношения, на которое ведет ссылка?

Здесь **существуют три подхода, каждый из которых поддерживает целостность по ссылкам.**

1. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа).
2. При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным.
3. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

В развитых реляционных СУБД обычно можно выбрать способ поддержания целостности по ссылкам для каждой отдельной ситуации определения внешнего ключа. Конечно, для принятия такого решения необходимо анализировать требования конкретной прикладной области.

Базисные средства манипулирования реляционными данными. Реляционное исчисление. Реляционная алгебра.

Существует три составляющих реляционной модели данных: структурная, целостная и манипуляционная.

В манипуляционной составляющей определяются два базовых механизма манипулирования реляционными данными - основанная на теории множеств реляционная алгебра и базирующееся на математической логике (точнее, на исчислении предикатов первого порядка) реляционное исчисление. В свою очередь, обычно рассматриваются два вида реляционного исчисления - исчисление доменов и исчисление предикатов.

Все эти механизмы обладают одним важным свойством: они замкнуты относительно понятия отношения. Это означает, что выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и результатом вычисления также являются отношения. В результате любое выражение или формула могут интерпретироваться как отношения, что позволяет использовать их в других выражениях или формулах.

Алгебра и исчисление обладают большой выразительной мощностью: очень сложные запросы к базе данных могут быть выражены с помощью одного выражения реляционной алгебры или

одной формулы реляционного исчисления. Именно по этой причине именно эти механизмы включены в реляционную модель данных.

Конкретный **язык манипулирования реляционными БД называется реляционно полным, если** любой запрос, выражаемый с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления, может быть выражен с помощью одного оператора этого языка.

Известно (и мы не будем это доказывать), что **механизмы реляционной алгебры и реляционного исчисления эквивалентны**, т.е. для любого допустимого выражения реляционной алгебры можно построить эквивалентную (т.е. производящую такой же результат) формулу реляционного исчисления и наоборот. Почему же в реляционной модели данных присутствуют оба эти механизма?

Дело в том, что они различаются уровнем процедурности. Выражения реляционной алгебры строятся на основе алгебраических операций (высокого уровня), и подобно тому, как интерпретируются арифметические и логические выражения, выражение реляционной алгебры также имеет процедурную интерпретацию. Другими словами, запрос, представленный на языке реляционной алгебры, может быть вычислен на основе вычисления элементарных алгебраических операций с учетом их старшинства и возможного наличия скобок. Для формулы реляционного исчисления однозначная интерпретация, вообще говоря, отсутствует. Формула только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения. Поэтому языки реляционного исчисления являются более непроцедурными или декларативными.

Поскольку механизмы реляционной алгебры и реляционного исчисления эквивалентны, то в конкретной ситуации для проверки степени реляционности некоторого языка БД можно пользоваться любым из этих механизмов.

Заметим, что крайне редко алгебра или исчисление принимаются в качестве полной основы какого-либо языка БД. Обычно (как, например, в случае языка SQL) язык основывается на некоторой смеси алгебраических и логических конструкций. Тем не менее, знание алгебраических и логических основ языков баз данных часто бывает полезно на практике.

Реляционная алгебра

Основная идея реляционной алгебры состоит в том, что коль скоро отношения являются множествами, то средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях, дополненных некоторыми специальными операциями, специфичными для баз данных.

Существует много подходов к определению реляционной алгебры, которые различаются набором операций и способами их интерпретации, но в принципе, более или менее равносильны.

Мы опишем немного расширенный **начальный вариант алгебры**, который был **предложен Коддом**. В этом варианте **набор основных алгебраических операций состоит из восьми операций, которые делятся на два класса** - теоретико-множественные операции и специальные реляционные операции.

Теоретико-множественные операции это:

- объединение отношений;
- пересечение отношений;
- взятие разности отношений;
- прямое произведение отношений.

Специальные реляционные операции это:

- ограничение отношения;
- проекция отношения;
- соединение отношений;
- деление отношений.

Кроме того, в состав алгебры включается **операция присваивания**, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и **операция переименования атрибутов**, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

*****Общая интерпретация реляционных операций

Если не вдаваться в некоторые тонкости, которые мы рассмотрим в следующих подразделах, то почти все операции предложенного выше набора обладают очевидной и простой интерпретацией.

- При выполнении операции объединения двух отношений производится отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов.
- Операция пересечения двух отношений производит отношение, включающее все кортежи, входящие в оба отношения-операнда.
- Отношение, являющееся разностью двух отношений включает все кортежи, входящие в отношение - первый операнд, такие, что ни один из них не входит в отношение, являющееся вторым операндом.
- При выполнении прямого произведения двух отношений производится отношение, кортежи которого являются конкатенацией (сцеплением) кортежей первого и второго операндов.
- Результатом ограничения отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.
- При выполнении проекции отношения на заданный набор его атрибутов производится отношение, кортежи которого производятся путем взятия соответствующих значений из кортежей отношения-операнда.
- При соединении двух отношений по некоторому условию образуется результирующее отношение, кортежи которого являются конкатенацией кортежей первого и второго отношений и удовлетворяют этому условию.
- У операции реляционного деления два операнда - бинарное и унарное отношения. Результирующее отношение состоит из одноатрибутных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) совпадает со множеством значений второго операнда.
- Операция переименования производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены.
- Операция присваивания позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

Поскольку результатом любой реляционной операции (кроме операции присваивания) является некоторое отношение, можно образовывать реляционные выражения, в которых вместо отношения-операнда некоторой реляционной операции находится вложенное реляционное выражение.

*****Замкнутость реляционной алгебры и операция переименования

Каждое отношение характеризуется схемой (или заголовком) и набором кортежей (или телом). Поэтому, если действительно желать иметь алгебру, операции которой замкнуты относительно понятия отношения, то каждая операция должна производить отношение в полном смысле, т.е. оно должно обладать и телом, и заголовком. Только в этом случае будет действительно возможно строить вложенные выражения.

Заголовок отношения представляет собой множество пар <имя-атрибута, имя-домена>. Если посмотреть на общий обзор реляционных операций, приведенный в предыдущем подразделе, то видно, что домены атрибутов результирующего отношения однозначно определяются доменами отношений-операндов. Однако с именами атрибутов результата не всегда все так просто.

Например, представим себе, что у отношений-операндов операции прямого произведения имеются одноименные атрибуты с одинаковыми доменами. Каким был бы заголовок результирующего отношения? Поскольку это множество, в нем не должны содержаться одинаковые элементы. Но и потерять атрибут в результате недопустимо. А это значит, что в этом случае вообще невозможно корректно выполнить операцию прямого произведения.

Аналогичные проблемы могут возникать и в случаях других двуместных операций. Для их разрешения в состав операций реляционной алгебры вводится операция переименования. Ее следует применять в любом случае, когда возникает конфликт именования атрибутов в отношениях - операндах одной реляционной операции. Тогда к одному из операндов сначала применяется операция переименования, а затем основная операция выполняется уже безо всяких проблем.

В дальнейшем изложении мы будем предполагать применение операции переименования во всех конфликтных случаях.

*****Особенности теоретико-множественных операций реляционной алгебры

Хотя в основе теоретико-множественной части реляционной алгебры лежит классическая теория множеств, соответствующие операции реляционной алгебры обладают некоторыми особенностями.

Начнем с операции объединения (все, что будет говориться по поводу объединения, переносится на операции пересечения и взятия разности). Смысл операции объединения в реляционной алгебре в целом остается теоретико-множественным. Но если в теории множеств операция объединения осмысленна для любых двух множеств-операндов, то в случае реляционной алгебры результатом операции объединения должно являться отношение. Если допустить в реляционной алгебре возможность теоретико-множественного объединения произвольных двух отношений (с разными схемами), то, конечно, результатом операции будет множество, но множество разнотипных кортежей, т.е. не отношение. Если исходить из требования замкнутости реляционной алгебры относительно понятия отношения, то такая операция объединения является бессмысленной.

Все эти соображения приводят к появлению понятия совместимости отношений по объединению: два отношения совместимы по объединению в том и только в том случае, когда обладают одинаковыми заголовками. Более точно, это означает, что в заголовках обоих отношений содержится один и тот же набор имен атрибутов, и одноименные атрибуты определены на одном и том же домене.

Если два отношения совместимы по объединению, то при обычном выполнении над ними операций объединения, пересечения и взятия разности результатом операции является отношение с корректно определенным заголовком, совпадающим с заголовком каждого из отношений-операндов. Напомним, что если два отношения "почти" совместимы по объединению, т.е. совместимы во всем, кроме имен атрибутов, то до выполнения операции типа соединения эти отношения можно сделать полностью совместимыми по объединению путем применения операции переименования.

Заметим, что включение в состав операций реляционной алгебры трех операций объединения, пересечения и взятия разности является очевидно избыточным, поскольку известно, что любая из этих операций выражается через две других. Тем не менее, Кодд в свое время решил включить все три операции, исходя из интуитивных потребностей потенциального пользователя системы реляционных БД, далекого от математики.

Другие проблемы связаны с операцией взятия прямого произведения двух отношений. В теории множеств прямое произведение может быть получено для любых двух множеств, и элементами результирующего множества являются пары, составленные из элементов первого и второго множеств. Поскольку отношения являются множествами, то и для любых двух отношений возможно получение прямого произведения. Но результат не будет отношением! Элементами результата будут являться не кортежи, а пары кортежей.

Поэтому в реляционной алгебре используется специализированная форма операции взятия прямого произведения - расширенное прямое произведение отношений. При взятии расширенного прямого произведения двух отношений элементом результирующего отношения является кортеж, являющийся конкатенацией (или слиянием) одного кортежа первого отношения и одного кортежа второго отношения.

Но теперь возникает второй вопрос - как получить корректно сформированный заголовок отношения-результата? Очевидно, что проблемой может быть именование атрибутов результирующего отношения, если отношения-операнды обладают одноименными атрибутами.

Эти соображения приводят к появлению **понятия совместимости по взятию расширенного прямого произведения.** Два отношения совместимы по взятию прямого произведения в том и только в том случае, если множества имен атрибутов этих отношений не пересекаются. Любые два отношения могут быть сделаны совместимыми по взятию прямого произведения путем применения операции переименования к одному из этих отношений.

Следует заметить, что операция взятия прямого произведения не является слишком осмысленной на практике. Во-первых, мощность ее результата очень велика даже при допустимых мощностях операндов, а во-вторых, результат операции не более информативен, чем взятые в совокупности операнды. Как мы увидим немного ниже, основной смысл включения операции расширенного прямого произведения в состав реляционной алгебры состоит в том, что на ее основе определяется действительно полезная операция соединения.

По поводу теоретико-множественных операций реляционной алгебры следует еще заметить, что все четыре операции являются ассоциативными. Т. е., если обозначить через OP любую из четырех операций, то (A OP B) OP C = A (B OP C), и следовательно, без введения двусмысленности можно писать A OP B OP C (A, B и C - отношения, обладающие свойствами,

требуемыми для корректного выполнения соответствующей операции). Все операции, кроме взятия разности, являются коммутативными, т.е. A OP B = B OP A.

*****Специальные реляционные операции

В этом подразделе мы несколько подробнее рассмотрим специальные реляционные операции реляционной алгебры: ограничение, проекция, соединение и деление.

Операция ограничения

Операция ограничения требует наличия двух операндов: ограничиваемого отношения и простого условия ограничения. Простое условие ограничения может иметь либо вид (a comp-op b), где a и b - имена атрибутов ограничиваемого отношения, для которых осмысленна операция сравнения comp-op, либо вид (a comp-op const), где a - имя атрибута ограничиваемого отношения, a const - литерально заданная константа.

В результате выполнения операции ограничения производится отношение, заголовок которого совпадает с заголовком отношения-операнда, а в тело входят те кортежи отношения-операнда, для которых значением условия ограничения является true.

Пусть UNION обозначает операцию объединения, INTERSECT - операцию пересечения, а MINUS - операцию взятия разности. Для обозначения операции ограничения будем использовать конструкцию A WHERE comp, где A - ограничиваемое отношение, a comp - простое условие сравнения. Пусть comp1 и comp2 - два простых условия ограничения. Тогда по определению:

- A WHERE comp1 AND comp2 обозначает то же самое, что и (A WHERE comp1) INTERSECT (A WHERE comp2)
- A WHERE comp1 OR comp2 обозначает то же самое, что и (A WHERE comp1) UNION (A WHERE comp2)
- A WHERE NOT comp1 обозначает то же самое, что и A MINUS (A WHERE comp1)

С использованием этих определений можно использовать операции ограничения, в которых условием ограничения является произвольное булевское выражение, составленное из простых условий с использованием логических связок AND, OR, NOT и скобок.

На интуитивном уровне операцию ограничения лучше всего представлять как взятие некоторой "горизонтальной" вырезки из отношения-операнда.

Операция взятия проекции

Операция взятия проекции также требует наличия двух операндов - проецируемого отношения A и списка имен атрибутов, входящих в заголовок отношения A.

Результатом проекции отношения A по списку атрибутов a_1, a_2, ..., a_n является отношение, с заголовком, определяемым множеством атрибутов a_1, a_2, ..., a_n, и с телом, состоящим из кортежей вида $<a_1:v_1, a_2:v_2, ..., a_n:v_n>$ таких, что в отношении A имеется кортеж, атрибут a_1 которого имеет значение v_1, атрибут a_2 имеет значение v_2, ..., атрибут a_n имеет значение v_n. Тем самым, при выполнении операции проекции выделяется "вертикальная" вырезка отношения-операнда с естественным уничтожением потенциально возникающих кортежей-дубликатов.

Операция соединения отношений

Общая операция соединения (называемая также соединением по условию) требует наличия двух операндов - соединяемых отношений и третьего операнда - простого условия. Пусть соединяются отношения A и B. Как и в случае операции ограничения, условие соединения comp имеет вид либо (a comp-op b), либо (a comp-op const), где a и b - имена атрибутов отношений A и B, const - литерально заданная константа, a comp-op - допустимая в данном контексте операция сравнения.

Тогда по определению результатом операции сравнения является отношение, получаемое путем выполнения операции ограничения по условию comp прямого произведения отношений A и B.

Если внимательно осмыслить это определение, то станет ясно, что в общем случае применение условия соединения существенно уменьшит мощность результата промежуточного прямого произведения отношений-операндов только в том случае, когда условие соединения имеет вид (a comp-op b), где a и b - имена атрибутов разных отношений-операндов. Поэтому на практике обычно считают реальными операциями соединения именно те операции, которые основываются на условии соединения приведенного вида.

Хотя операция соединение в нашей интерпретации не является примитивной (поскольку она определяется с использованием прямого произведения и проекции), в силу особой практической важности она включается в базовый набор операций реляционной алгебры. Заметим также, что в практических реализациях соединение обычно не выполняется именно как ограничение прямого

произведения. Имеются более эффективные алгоритмы, гарантирующие получение такого же результата.

Имеется важный частный случай соединения - эквисоединение и простое, но важное расширение операции эквисоединения - естественное соединение. Операция соединения называется *операцией эквисоединения*, если условие соединения имеет вид (a = b), где a и b - атрибуты разных операндов соединения. Этот случай важен потому, что (a) он часто встречается на практике, и (b) для него существуют эффективные алгоритмы реализации.

Операция естественного соединения применяется к паре отношений A и B, обладающих (возможно составным) общим атрибутом c (т.е. атрибутом с одним и тем же именем и определенным на одном и том же домене). Пусть ab обозначает объединение заголовков отношений A и B. Тогда естественное соединение A и B - это спроектированный на ab результат эквисоединения A и B по A/c и BBC. Если вспомнить введенное нами в конце предыдущей главы определение внешнего ключа отношения, то должно стать понятно, что основной смысл операции естественного соединения - возможность восстановления сложной сущности, декомпозированной по причине требования первой нормальной формы. Операция естественного соединения не включается прямо в состав набора операций реляционной алгебры, но она имеет очень важное практическое значение.

Операция деления отношений

Эта операция наименее очевидна из всех операций реляционной алгебры и поэтому нуждается в более подробном объяснении. Пусть заданы два отношения - A с заголовком $\{a_1, a_2, ..., a_n, b_1, b_2, ..., b_m\}$ и B с заголовком $\{b_1, b_2, ..., b_m\}$. Будем считать, что атрибут b_i отношения A и атрибут b_i отношения B не только обладают одним и тем же именем, но и определены на одном и том же домене. Назовем множество атрибутов $\{a_j\}$ составным атрибутом a, а множество атрибутов $\{b_j\}$ - составным атрибутом b. После этого будем говорить о реляционном делении бинарного отношения A(a,b) на унарное отношение B(b).

Результатом деления A на B является унарное отношение C(a), состоящее из кортежей v таких, что в отношении A имеются кортежи <v, w> такие, что множество значений {w} включает множество значений атрибута b в отношении B.

Предположим, что в базе данных сотрудников поддерживаются два отношения: СОТРУДНИКИ (ИМЯ, ОТД_НОМЕР) и ИМЕНА (ИМЯ), причем унарное отношение ИМЕНА содержит все фамилии, которыми обладают сотрудники организации. Тогда после выполнения операции реляционного деления отношения СОТРУДНИКИ на отношение ИМЕНА будет получено унарное отношение, содержащее номера отделов, сотрудники которых обладают всеми возможными в этой организации именами.

Реляционное исчисление

Предположим, что мы работаем с базой данных, обладающей схемой СОТРУДНИКИ (СОТР_НОМ, СОТР_ИМЯ, СОТР_ЗАРП, ОТД_НОМ) и ОТДЕЛЫ (ОТД_НОМ, ОТД_КОЛ, ОТД_НАЧ), и хотим узнать имена и номера сотрудников, являющихся начальниками отделов с количеством сотрудников больше 50.

Если бы для формулировки такого запроса использовалась реляционная алгебра, то мы получили бы алгебраическое выражение, которое читалось бы, например, следующим образом:

- выполнить соединение отношений СОТРУДНИКИ и ОТДЕЛЫ по условию СОТР_НОМ = ОТД_НАЧ;
- ограничить полученное отношение по условию ОТД_КОЛ > 50;
- спроецировать результат предыдущей операции на атрибут СОТР_ИМЯ, СОТР_НОМ.

Мы четко сформулировали последовательность шагов выполнения запроса, каждый из которых соответствует одной реляционной операции. Если же сформулировать тот же запрос с использованием реляционного исчисления, которому посвящается этот раздел, то мы получили бы формулу, которую можно было бы прочитать, например, следующим образом: Выдать СОТР_ИМЯ и СОТР_НОМ для сотрудников таких, что существует отдел с таким же значением ОТД_НАЧ и значением ОТД_КОЛ большим 50.

Во второй формулировке мы указали лишь характеристики результирующего отношения, но ничего не сказали о способе его формирования. В этом случае система должна сама решить, какие операции и в каком порядке нужно выполнить над отношениями СОТРУДНИКИ и ОТДЕЛЫ. Обычно говорят, что алгебраическая формулировка является процедурной, т.е. задающей правила выполнения запроса, а логическая - описательной (или декларативной), поскольку она всего лишь описывает свойства желаемого результата. Как мы указывали в начале лекции, на самом деле эти

два механизма эквивалентны и существуют не очень сложные правила преобразования одного формализма в другой.

*****Кортежные переменные и правильно построенные формулы

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка. Базисными понятиями исчисления являются понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы.

В зависимости от того, что является областью определения переменной, различаются исчисление кортежей и исчисление доменов. В исчислении кортежей областями определения переменных являются отношения базы данных, т.е. допустимым значением каждой переменной является кортеж некоторого отношения. В исчислении доменов областями определения переменных являются домены, на которых определены атрибуты отношений базы данных, т.е. допустимым значением каждой переменной является значение некоторого домена. Мы рассмотрим более подробно исчисление кортежей, а в конце лекции коротко опишем особенности исчисления доменов.

В отличие от раздела, посвященного реляционной алгебре, в этом разделе нам не удастся избежать использования некоторого конкретного синтаксиса, который мы, тем не менее, формально определять не будем. Необходимые синтаксические конструкции будут вводиться по мере необходимости. В совокупности, используемый синтаксис близок, но не полностью совпадает с синтаксисом языка баз данных QUEL, который долгое время являлся основным языком СУБД Ingres.

Для определения кортежной переменной используется оператор RANGE. Например, для того, чтобы определить переменную СОТРУДНИК, областью определения которой является отношение СОТРУДНИКИ, нужно употребить конструкцию

RANGE СОТРУДНИК IS СОТРУДНИКИ

Как мы уже говорили, из этого определения следует, что в любой момент времени переменная СОТРУДНИК представляет некоторый кортеж отношения СОТРУДНИКИ. При использовании кортежных переменных в формулах можно ссылаться на значение атрибута переменной (это аналогично тому, как, например, при программировании на языке Си можно сослаться на значение поля структурной переменной). Например, для того, чтобы сослаться на значение атрибута СОТР_ИМЯ переменной СОТРУДНИК, нужно употребить конструкцию СОТРУДНИК.СОТР_ИМЯ.

Правильно построенные формулы (WFF - Well-Formed Formula) служат для выражения условий, накладываемых на кортежные переменные. Основой WFF являются простые сравнения (comparison), представляющие собой операции сравнения скалярных значений (значений атрибутов переменных или литерально заданных констант). Например, конструкция "СОТРУДНИК.СОТР_НОМ = 140" является простым сравнением. По определению, простое сравнение является WFF, а WFF, заключенная в круглые скобки, является простым сравнением.

Более сложные варианты WFF строятся с помощью логических связок NOT, AND, OR и IF ... THEN. Так, если form - WFF, а comp - простое сравнение, то NOT form, comp AND form, comp OR form и IF comp THEN form являются WFF.

Наконец, допускается построение WFF с помощью кванторов. Если form - это WFF, в которой участвует переменная var, то конструкции EXISTS var (form) и FORALL var (form) представляют wff.

Переменные, входящие в WFF, могут быть свободными или связанными. Все переменные, входящие в WFF, при построении которой не использовались кванторы, являются свободными. Фактически, это означает, что если для какого-то набора значений свободных кортежных переменных при вычислении WFF получено значение true, то эти значения кортежных переменных могут входить в результирующее отношение. Если же имя переменной использовано сразу после квантора при построении WFF вида EXISTS var (form) или FORALL var (form), то в этой WFF и во всех WFF, построенных с ее участием, var - это связанная переменная. Это означает, что такая переменная не видна за пределами минимальной WFF, связавшей эту переменную. При вычислении значения такой WFF используется не одно значение связанной переменной, а вся ее область определения.

Пусть СОТР1 и СОТР2 - две кортежные переменные, определенные на отношении СОТРУДНИКИ. Тогда, WFF EXISTS СОТР2 (СОТР1.СОТР_ЗАРП > СОТР2.СОТР_ЗАРП) для текущего кортежа переменной СОТР1 принимает значение true в том и только в том случае, если во всем отношении СОТРУДНИКИ найдется кортеж (связанный с переменной СОТР2) такой, что

значение его атрибута СОТР_ЗАРП удовлетворяет внутреннему условию сравнения. WFF FORALL СОТР2 (СОТР1.СОТР_ЗАРП > СОТР2.СОТР_ЗАРП) для текущего кортежа переменной СОТР1 принимает значение true в том и только в том случае, если для всех кортежей отношения СОТРУДНИКИ (связанных с переменной СОТР2) значения атрибута СОТР_ЗАРП удовлетворяют условию сравнения.

На самом деле, правильнее говорить не о свободных и связанных переменных, а о свободных и связанных вхождениях переменных. Легко видеть, что если переменная var является связанной в WFF form, то во всех WFF, включающих данную, может использоваться имя переменной var, которая может быть свободной или связанной, но в любом случае не имеет никакого отношения к вхождению переменной var в WFF form. Вот пример:

EXISTS СОТР2 (СОТР1.СОТР_ОТД_НОМ = СОТР2.СОТР_ОТД_НОМ) AND
FORALL СОТР2 (СОТР1.СОТР_ЗАРП > СОТР2.СОТР_ЗАРП)

Здесь мы имеем два связанных вхождения переменной СОТР2 с совершенно разным смыслом.

*****Целевые списки и выражения реляционного исчисления

Итак, WFF обеспечивают средства формулировки условия выборки из отношений БД. Чтобы можно было использовать исчисление для реальной работы с БД, требуется еще один компонент, который определяет набор и имена столбцов результирующего отношения. Этот компонент называется целевым списком (target_list).

Целевой список строится из целевых элементов, каждый из которых может иметь следующий вид:

- var.attr, где var - имя свободной переменной соответствующей WFF, а attr - имя атрибута отношения, на котором определена переменная var;
- var, что эквивалентно наличию подсписка var.attr1, var.attr2, ..., var.attrn, где attr1, attr2, ..., attrn включает имена всех атрибутов определяющего отношения;
- new_name = var.attr; new_name - новое имя соответствующего атрибута результирующего отношения.

Последний вариант требуется в тех случаях, когда в WFF используются несколько свободных переменных с одинаковой областью определения.

Выражением реляционного исчисления кортежей называется конструкция вида target_list WHERE wff. Значением выражения является отношение, тело которого определяется WFF, а набор атрибутов и их имена - целевым списком.

*****Реляционное исчисление доменов

В исчислении доменов областью определения переменных являются не отношения, а домены. Применительно к базе данных СОТРУДНИКИ-ОТДЕЛЫ можно говорить, например, о доменных переменных ИМЯ (значения - допустимые имена) или НОСОТР (значения - допустимые номера сотрудников).

Основным формальным отличием исчисления доменов от исчисления кортежей является наличие дополнительного набора предикатов, позволяющих выражать так называемые условия членства. Если R - это n-арное отношение с атрибутами a_1, a_2, ..., a_n, то условие членства имеет вид

R (ai1:vi1, ai2:vi2, ..., aim:vim) (m <= n),

где v_{ij} - это либо литерально задаваемая константа, либо имя кортежной переменной. Условие членства принимает значение true в том и только в том случае, если в отношении R существует кортеж, содержащий указанные значения указанных атрибутов. Если v_{ij} - константа, то на атрибут a_{ij} задается жесткое условие, не зависящее от текущих значений доменных переменных; если же v_{ij} - имя доменной переменной, то условие членства может принимать разные значения при разных значениях этой переменной.

Во всех остальных отношениях формулы и выражения исчисления доменов выглядят похожими на формулы и выражения исчисления кортежей. В частности, конечно, различаются свободные и связанные вхождения доменных переменных.

Для примера сформулируем с использованием исчисления доменов запрос "Выдать номера и имена сотрудников, не получающих минимальную заработную плату" (будем считать для простоты, что мы определили доменные переменные, имена которых совпадают с именами атрибутов отношения СОТРУДНИКИ, а в случае, когда требуется несколько доменных переменных, определенных на одном домене, мы будем добавлять в конце имени цифры):

СОТР_НОМ, СОТР_ИМЯ WHERE EXISTS СОТР_ЗАРП1
(СОТРУДНИКИ (СОТР_ЗАРП1) AND

СОТРУДНИКИ (СОТР_НОМ, СОТР_ИМЯ, СОТР_ЗАРП) AND
СОТР_ЗАРП > СОТР_ЗАРП1)

Реляционное исчисление доменов является основой большинства языков запросов, основанных на использовании форм. В частности, на этом исчислении базировался известный язык Query-by-Example, который был первым (и наиболее интересным) языком в семействе языков, основанных на табличных формах.

Проектирование баз данных. Нормализация отношений. Функциональная зависимость. Реляционная полнота.

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. 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.

При проектировании базы данных решаются две основных проблемы:

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

В случае реляционных баз данных трудно представить какие-либо общие рецепты по части физического проектирования. Здесь слишком много зависит от используемой СУБД. Поэтому мы ограничимся вопросами логического проектирования реляционных баз данных, которые существенны при использовании любой реляционной СУБД.

Более того, мы не будем касаться очень важного аспекта проектирования - определения ограничений целостности (за исключением ограничения первичного ключа). Дело в том, что при использовании СУБД с развитыми механизмами ограничений целостности (например, SQL-ориентированных систем) трудно предложить какой-либо общий подход к определению ограничений целостности. Эти ограничения могут иметь очень общий вид, и их формулировка пока относится скорее к области искусства, чем инженерного мастерства. Самое большее, что предлагается по этому поводу в литературе, это автоматическая проверка непротиворечивости набора ограничений целостности.

Так что будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том,

- из каких отношений должна состоять БД и
- какие атрибуты должны быть у этих отношений.

*****О нормализации, функциональных и многозначных зависимостях

Нормализация – это разбиение таблицы на две или более, обладающих лучшими свойствами при включении, изменении и удалении данных. Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором *каждый факт появляется лишь в одном месте*, т.е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных.

Каждая таблица в реляционной БД удовлетворяет условию, в соответствии с которым в позиции на пересечении каждой строки и столбца таблицы всегда находится единственное атомарное значение, и никогда не может быть множества таких значений. Любая таблица, удовлетворяющая этому условию, называется нормализованной. Фактически, ненормализованные таблицы, т.е. таблицы, содержащие повторяющиеся группы, даже не допускаются в реляционной БД.

Всякая нормализованная таблица автоматически считается таблицей в *первой нормальной форме*, сокращенно *1НФ*. Таким образом, строго говоря, "нормализованная" и "находящаяся в 1НФ" означают одно и то же. Однако на практике термин "нормализованная" часто используется в более узком смысле – "полностью нормализованная", который означает, что в проекте не нарушаются никакие принципы нормализации.

Таблица находится в первой нормальной форме (1НФ) тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто.

Теперь в дополнение к 1НФ можно определить дальнейшие уровни нормализации – *вторую нормальную форму* (*2НФ*), *третью нормальную форму* (*3НФ*) и т.д. По существу, таблица находится в 2НФ, если она находится в 1НФ и удовлетворяет, кроме того, некоторому дополнительному условию, суть которого будет рассмотрена ниже. Таблица находится в 3НФ, если она находится в 2НФ и, помимо этого, удовлетворяет еще другому дополнительному условию и т.д.

Таким образом, каждая нормальная форма является в некотором смысле более ограниченной, но и более *желательной*, чем предшествующая. Это связано с тем, что "(N+1)-я нормальная форма" не обладает некоторыми непривлекательными особенностями, свойственным "N-й нормальной форме". Общий смысл дополнительного условия, налагаемого на (N+1)-ю нормальную форму по отношению к N-й нормальной форме, состоит в исключении этих непривлекательных особенностей.

Теория нормализации основывается на наличии той или иной зависимости между полями таблицы. Определены два вида таких зависимостей: функциональные и многозначные.

Функциональная зависимость. Поле В таблицы функционально зависит от поля А той же таблицы в том и только в том случае, когда в любой заданный момент времени для каждого из

различных значений поля A обязательно существует только одно из различных значений поля B. Отметим, что здесь допускается, что поля A и B могут быть составными.

Например, в таблице Блюда (рис. 4.4)

Поставщики

Поставщик	Город	Страна
"Полесье"	Киев	Украина
"Наталка"	Киев	Украина
"Хуанхэ"	Пекин	Китай
"Лайма"	Рига	Латвия
"Юрмала"	Рига	Латвия
...	...	...

Рис 4.3

Блюда

БЛ	Блюдо	Вид
1	Лобио	Закуска
2	Харчо	Суп
3	Шашлык	Горячее
4	Кофе	Десерт
...	...	...

Рис 4.4

поля Блюдо и Вид функционально зависят от ключа БЛ, а в таблице Поставщики рис. 4.3 поле Страна функционально зависит от составного ключа (Поставщик, Город). Однако последняя зависимость не является функционально полной, так как Страна функционально зависит и от части ключа – поля Город.

Полная функциональная зависимость. Поле В находится в полной функциональной зависимости от составного поля А, если оно функционально зависит от А и не зависит функционально от любого подмножества поля А.

Многозначная зависимость. Поле А многозначно определяет поле В той же таблицы, если для каждого значения поля А существует хорошо определенное множество соответствующих значений В.

Обучение

Дисциплина	**Преподаватель**	**Учебник**
Информатика	Шипилов П.А.	Форсайт Р. Паскаль для всех
Информатика	Шипилов П.А.	Уэйт М. и др. Язык Си
Информатика	Голованевский Г.Л.	Форсайт Р. Паскаль для всех
Информатика	Голованевский Г.Л.	Уэйт М. и др. Язык Си
...	...	...

Рис. 4.5. К иллюстрации многозначных зависимостей

Для примера рассмотрим таблицу "Обучение" (рис. 4.5). В ней есть многозначная зависимость "Дисциплина-Преподаватель": дисциплина (в примере Информатика) может может читаться несколькими преподавателями (в примере Шипиловым и Голованевским). Есть и другая многозначная зависимость "Дисциплина-Учебник": при изучении Информатики используются учебники "Паскаль для всех" и "Язык Си". При этом Преподаватель и Учебник не связныфункциональной зависимостью, что приводит к появлению избыточности (для добавление еще одного учебника придется ввести в таблицу две новых строки). Дело улучшается при замене этой таблицы на две: (Дисциплина-Преподаватель и Дисциплина-Учебник).

Проектирование реляционных баз данных с использованием нормализации

Сначала будет рассмотрен классический подход, при котором весь процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами. Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы - значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требованию.

В теории реляционных баз данных обычно выделяется следующая **последовательность нормальных форм:**

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

1. каждая следующая нормальная форма в некотором смысле лучше предыдущей;
2. при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

В основе процесса проектирования лежит метод нормализации, декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии *функциональной зависимости*. Для дальнейшего изложения нам потребуются несколько определений.

Определение 1. Функциональная зависимость

В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: R.X (r) R.Y.

Определение 2. Полная функциональная зависимость

Функциональная зависимость R.X (r) R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

Определение 3. Транзитивная функциональная зависимость

Функциональная зависимость R.X −> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X −> R.Z и R.Z −> R.Y и отсутствует функциональная зависимость R.Z --> R.X. (При отсутствии последнего требования мы имели бы "неинтересные" транзитивные зависимости в любом отношении, обладающем несколькими ключами.)

Определение 4. Неключевой атрибут

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа (в частности, первичного).

Определение 5. Взаимно независимые атрибуты

Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

*****Вторая нормальная форма

Рассмотрим следующий пример схемы отношения:
СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ
(СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)
Первичный ключ:
СОТР_НОМЕР, ПРО_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР −> СОТР_ЗАРП
СОТР_НОМЕР −> ОТД_НОМЕР
ОТД_НОМЕР −> СОТР_ЗАРП
СОТР_НОМЕР, ПРО_НОМЕР −> СОТР_ЗАДАН

Как видно, хотя первичным ключом является составной атрибут СОТР_НОМЕР, ПРО_НОМЕР, атрибуты СОТР_ЗАРП и ОТД_НОМЕР функционально зависят от части первичного ключа, атрибута СОТР_НОМЕР. В результате мы не сможем вставить в отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ кортеж, описывающий сотрудника, который еще не выполняет никакого проекта (первичный ключ не может содержать неопределенное значение). При удалении кортежа мы не только разрушаем связь данного сотрудника с данным проектом, но утрачиваем информацию о том, что он работает в некотором отделе. При переводе сотрудника в другой отдел мы будем вынуждены модифицировать все кортежи, описывающие этого сотрудника, или получим несогласованный результат. Такие неприятные явления называются аномалиями схемы отношения. Они устраняются путем нормализации.

Определение 6. Вторая нормальная форма (в этом определении предполагается, что единственным ключом отношения является первичный ключ)

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда находится в 1NF, и каждый неключевой атрибут полностью зависит от первичного ключа.

Можно произвести следующую декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ в два отношения СОТРУДНИКИ-ОТДЕЛЫ и СОТРУДНИКИ-ПРОЕКТЫ:

СОТРУДНИКИ-ОТДЕЛЫ (СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР)
Первичный ключ:

СОТР_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР −> СОТР_ЗАРП
СОТР_НОМЕР −> ОТД_НОМЕР
ОТД_НОМЕР −> СОТР_ЗАРП
СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)
Первичный ключ:
СОТР_НОМЕР, ПРО_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР, ПРО_НОМЕР −> СОТР_ЗАДАН

Каждое из этих двух отношений находится в 2NF, и в них устранены отмеченные выше аномалии (легко проверить, что все указанные операции выполняются без проблем).

Если допустить наличие нескольких ключей, то определение 6 примет следующий вид:

Определение 6 (2)

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда оно находится в 1NF, и каждый неключевой атрибут полностью зависит от каждого ключа R.

Здесь и далее мы не будем приводить примеры для отношений с несколькими ключами. Они слишком громоздки и относятся к ситуациям, редко встречающимся на практике.

*****Третья нормальная форма

Рассмотрим еще раз отношение СОТРУДНИКИ-ОТДЕЛЫ, находящееся в 2NF. Заметим, что функциональная зависимость СОТР_НОМЕР −> СОТР_ЗАРП является транзитивной; она является следствием функциональных зависимостей СОТР_НОМЕР −> ОТД_НОМЕР и ОТД_НОМЕР −> СОТР_ЗАРП. Другими словами, заработная плата сотрудника на самом деле является характеристикой не сотрудника, а отдела, в котором он работает (это не очень естественное предположение, но достаточное для примера).

В результате мы не сможем занести в базу данных информацию, характеризующую заработную плату отдела, до тех пор, пока в этом отделе не появится хотя бы один сотрудник (первичный ключ не может содержать неопределенное значение). При удалении кортежа, описывающего последнего сотрудника данного отдела, мы лишимся информации о заработной плате отдела. Чтобы согласованным образом изменить заработную плату отдела, мы будем вынуждены предварительно найти все кортежи, описывающие сотрудников этого отдела. Т.е. в отношении СОТРУДИКИ-ОТДЕЛЫ по-прежнему существуют аномалии. Их можно устранить путем дальнейшей нормализации.

Определение 7. Третья нормальная форма. (Снова определение дается в предположении существования единственного ключа.)

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Можно произвести декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ в два отношения СОТРУДНИКИ и ОТДЕЛЫ:

СОТРУДНИКИ (СОТР_НОМЕР, ОТД_НОМЕР)
Первичный ключ:
СОТР_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР −> ОТД_НОМЕР
ОТДЕЛЫ (ОТД_НОМЕР, СОТР_ЗАРП)
Первичный ключ:
ОТД_НОМЕР
Функциональные зависимости:
ОТД_НОМЕР −> СОТР_ЗАРП

Каждое из этих двух отношений находится в 3NF и свободно от отмеченных аномалий.

Если отказаться от того ограничения, что отношение обладает единственным ключом, то определение 3NF примет следующую форму:

Определение 7 (2)

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 1NF, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

*****Нормальная форма Бойса-Кодда

Рассмотрим следующий пример схемы отношения:
СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, СОТР_ИМЯ, ПРО_НОМЕР, СОТР_ЗАДАН)
Возможные ключи:
СОТР_НОМЕР, ПРО_НОМЕР
СОТР_ИМЯ, ПРО_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР −> СОТР_ИМЯ
СОТР_НОМЕР −> ПРО_НОМЕР
СОТР_ИМЯ −> СОТР_НОМЕР
СОТР_ИМЯ −> ПРО_НОМЕР
СОТР_НОМЕР, ПРО_НОМЕР −> СОТР_ЗАДАН
СОТР_ИМЯ, ПРО_НОМЕР −> СОТР_ЗАДАН

В этом примере мы предполагаем, что личность сотрудника полностью определяется как его номером, так и именем (это снова не очень жизненное предположение, но достаточное для примера).

В соответствии с определением 7~ отношение СОТРУДНИКИ-ПРОЕКТЫ находится в 3NF. Однако тот факт, что имеются функциональные зависимости атрибутов отношения от атрибута, являющегося частью первичного ключа, приводит к аномалиям. Например, для того, чтобы изменить имя сотрудника с данным номером согласованным образом, нам потребуется модифицировать все кортежи, включающие его номер.

Определение 8. Детерминант

Детерминант - любой атрибут, от которого полностью функционально зависит некоторый другой атрибут.

Определение 9. Нормальная форма Бойса-Кодда

Отношение R находится в нормальной форме Бойса-Кодда (BCNF) в том и только в том случае, если каждый детерминант является возможным ключом.

Очевидно, что это требование не выполнено для отношения СОТРУДНИКИ-ПРОЕКТЫ. Можно произвести его декомпозицию к отношениям СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ:
СОТРУДНИКИ (СОТР_НОМЕР, СОТР_ИМЯ)
Возможные ключи:
СОТР_НОМЕР
СОТР_ИМЯ
Функциональные зависимости:
СОТР_НОМЕР −> СОТР_ИМЯ
СОТР_ИМЯ −> СОТР_НОМЕР
СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)
Возможный ключ:
СОТР_НОМЕР, ПРО_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР, ПРО_НОМЕР −> СОТР_ЗАДАН

Возможна альтернативная декомпозиция, если выбрать за основу СОТР_ИМЯ. В обоих случаях получаемые отношения СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ находятся в BCNF, и им не свойственны отмеченные аномалии.

*****Четвертая нормальная форма

Рассмотрим пример следующей схемы отношения:
ПРОЕКТЫ (ПРО_НОМЕР,ПРО_СОТР, ПРО_ЗАДАН)

Отношение ПРОЕКТЫ содержит номера проектов, для каждого проекта список сотрудников, которые могут выполнять проект, и список заданий, предусматриваемых проектом. Сотрудники могут участвовать в нескольких проектах, и разные проекты могут включать одинаковые задания.

Каждый кортеж отношения связывает некоторый проект с сотрудником, участвующим в этом проекте, и заданием, который сотрудник выполняет в рамках данного проекта (мы предполагаем, что любой сотрудник, участвующий в проекте, выполняет все задания, предусмотренные этим проектом). По причине сформулированных выше условий единственным возможным ключем отношения является составной атрибут ПРО_НОМЕР, ПРО_СОТР, ПРО_ЗАДАН, и нет никаких других детерминантов. Следовательно, отношение ПРОЕКТЫ находится в BCNF. Но при этом оно обладает недостатками: если, например, некоторый сотрудник присоединяется к данному проекту, необходимо вставить в отношение ПРОЕКТЫ столько кортежей, сколько заданий в нем предусмотрено.

Определение 10. Многозначные зависимости

В отношении R (A, B, C) существует многозначная зависимость R.A −> −> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

В отношении ПРОЕКТЫ существуют следующие две многозначные зависимости:

ПРО_НОМЕР −> −> ПРО_СОТР

ПРО_НОМЕР −> −> ПРО_ЗАДАН

Легко показать, что в общем случае в отношении R (A, B, C) существует многозначная зависимость R.A −> −> R.B в том и только в том случае, когда существует многозначная зависимость R.A −> −> R.C.

Дальнейшая нормализация отношений, подобных отношению ПРОЕКТЫ, основывается на следующей теореме:

Теорема Фейджина

Отношение R (A, B, C) можно спроецировать без потерь в отношения R1 (A, B) и R2 (A, C) в том и только в том случае, когда существует MVD A −> −> B | C.

Под проецированием без потерь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений.

Определение 11. Четвертая нормальная форма

Отношение R находится в четвертой нормальной форме (4NF) в том и только в том случае, если в случае существования многозначной зависимости A −> −> B все остальные атрибуты R функционально зависят от A.

В нашем примере можно произвести декомпозицию отношения ПРОЕКТЫ в два отношения ПРОЕКТЫ-СОТРУДНИКИ и ПРОЕКТЫ-ЗАДАНИЯ:

ПРОЕКТЫ-СОТРУДНИКИ (ПРО_НОМЕР, ПРО_СОТР)

ПРОЕКТЫ-ЗАДАНИЯ (ПРО_НОМЕР, ПРО_ЗАДАН)

Оба эти отношения находятся в 4NF и свободны от отмеченных аномалий.

*****Пятая нормальная форма

Во всех рассмотренных до этого момента нормализациях производилась декомпозиция одного отношения в два. Иногда это сделать не удается, но возможна декомпозиция в большее число отношений, каждое из которых обладает лучшими свойствами.

Рассмотрим, например, отношение

СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ (СОТР_НОМЕР, ОТД_НОМЕР, ПРО_НОМЕР)

Предположим, что один и тот же сотрудник может работать в нескольких отделах и работать в каждом отделе над несколькими проектами. Первичным ключем этого отношения является полная совокупность его атрибутов, отсутствуют функциональные и многозначные зависимости.

Поэтому отношение находится в 4NF. Однако в нем могут существовать аномалии, которые можно устранить путем декомпозиции в три отношения.

Определение 12. Зависимость соединения

Отношение R (X, Y, ..., Z) удовлетворяет зависимости соединения * (X, Y, ..., Z) в том и только в том случае, когда R восстанавливается без потерь путем соединения своих проекций на X, Y, ..., Z.

Определение 13. Пятая нормальная форма

Отношение R находится в пятой нормальной форме (нормальной форме проекции-соединения - PJ/NF) в том и только в том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.

Введем следующие имена составных атрибутов:

СО = {СОТР_НОМЕР, ОТД_НОМЕР}

СП = {СОТР_НОМЕР, ПРО_НОМЕР}

ОП = {ОТД_НОМЕР, ПРО_НОМЕР}

Предположим, что в отношении СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ существует зависимость соединения:

* (СО, СП, ОП)

На примерах легко показать, что при вставках и удалениях кортежей могут возникнуть проблемы. Их можно устранить путем декомпозиции исходного отношения в три новых отношения:

СОТРУДНИКИ-ОТДЕЛЫ (СОТР_НОМЕР, ОТД_НОМЕР)

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР)

ОТДЕЛЫ-ПРОЕКТЫ (ОТД_НОМЕР, ПРО_НОМЕР)

Пятая нормальная форма - это последняя нормальная форма, которую можно получить путем декомпозиции. Ее условия достаточно нетривиальны, и на практике 5NF не используется. Заметим, что зависимость соединения является обобщением как многозначной зависимости, так и функциональной зависимости.

Процедура проектирования

Процесс проектирования информационных систем является достаточно сложной задачей. Он начинается с построения инфологической модели данных, т.е. идентификации сущностей. Затем необходимо выполнить следующие шаги процедуры проектирования даталогической модели.

1. Представить каждый стержень (независимую сущность) таблицей базы данных (базовой таблицей) и специфицировать первичный ключ этой базовой таблицы.

2. Представить каждую ассоциацию (связь вида "многие-ко-многим" или "многие-ко-многим-ко-многим" и т.д. между сущностями) как базовую таблицу. Использовать в этой таблице внешние ключи для идентификации участников ассоциации и специфицировать ограничения, связанные с каждым из этих внешних ключей.

3. Представить каждую характеристику как базовую таблицу с внешним ключом, идентифицирующим сущность, описываемую этой характеристикой. Специфицировать ограничения на внешний ключ этой таблицы и ее первичный ключ – по всей вероятности, комбинации этого внешнего ключа и свойства, которое гарантирует "уникальность в рамках описываемой сущности".

4. Представить каждое обозначение, которое не рассматривалось в предыдущем пункте, как базовую таблицу с внешним ключом, идентифицирующим обозначаемую сущность. Специфицировать связанные с каждым таким внешним ключом ограничения.

5. Представить каждое свойство как поле в базовой таблице, представляющей сущность, которая непосредственно описывается этим свойством.

6. Для того чтобы исключить в проекте непреднамеренные нарушения каких-либо принципов нормализации, выполнить процедуру нормализации.

7. Если в процессе нормализации было произведено разделение каких-либо таблиц, то следует модифицировать инфологическую модель базы данных и повторить перечисленные шаги.

8. Указать ограничения целостности проектируемой базы данных и дать (если это необходимо) краткое описание полученных таблиц и их полей.

Определение 1NF, 2NF, 3NF и проектирование БД простыми словами

Проектирование реляционной БД заключается в разработке структуры данных, т.е. в определении состава таблиц и связей между ними. При этом структура должна быть эффективной и обеспечивать:

- быстрый доступ к данным;
- отсутствие дублирования (повторения) данных

- целостность данных.

Проектирование БД можно представить следующим образом:

1. Сбор всей информации об объектах решаемой задачи в рамках одной таблицы (одного **отношения**)
2. Разбиение полученной таблицы на несколько взаимосвязанных таблиц на основе принципа нормализации отношений.

Для того, чтобы таблица находилась в первой нормальной форме, она должна соответствовать следующим 2 условиям:

1. Любое поле таблицы содержит неделимую информацию

(этот признак относителен и будет зависеть от того, как используется хранящаяся в таблице информация: можно сказать, что информация поля неделима, если нигде и не при каких операциях работы с данными не возникает необходимость выделять какую-то часть (фрагмент) хранящейся в нем информации)

2. В таблице отсутствуют повторяющиеся группы полей

Вторая нормальная форма требует следующего:

1. Таблица должна удовлетворять 1NF
2. Любое неключевое поле должно однозначно идентифицироваться ключевыми полями.

Третья нормальная форма:

1. Таблица должна удовлетворять 2NF
2. Ни одно из неключевыз полей не должно однозначно идентифицироваться значением другого неключевого поля (полей).

Еще одним важным правилом при проектировании базы данных является следующее:

- в таблицах базы данных не должно быть полей, хранящих такие значения, которые на самом деле могут быть в любой момент времени вычислены на основании значений других полей.

От этого правила следует отступать только в отдельных случаях, например,

- когда объем БД велик, а вычислительные мощности ЭВМ не позволяют обрабатывать его с приемлемой для пользователя скоростью,
- когда такие данные запрашиваются очень часто,
- когда связанные друг с другом данные различных полей вносятся в БД разными операторами и такое косвенное дублирование данных позволяет проверять вносимую информацию и предотвращать ошибки.

Графическое представление моделей данных. Диаграммы «сущность-связь». Отображение инфологической схемы на модели данных. Метод декомпозиции. Метод синтеза.

Семантическое моделирование данных, ER-диаграммы

Широкое распространение реляционных СУБД и их использование в самых разнообразных приложениях показывает, что реляционная модель данных достаточна для моделирования предметных областей. Однако проектирование реляционной базы данных в терминах отношений на основе кратко рассмотренного нами механизма нормализации часто представляет собой очень сложный и неудобный для проектировщика процесс.

При этом проявляется ограниченность реляционной модели данных в следующих аспектах:

- Модель не предоставляет достаточных средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика. В частности, это относится к упоминавшейся нами проблеме представления ограничений целостности.
- Для многих приложений трудно моделировать предметную область на основе плоских таблиц. В ряде случаев на самой начальной стадии проектирования проектировщику приходится производить насилие над собой, чтобы описать предметную область в виде одной (возможно, даже ненормализованной) таблицы.

- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет каких-либо средств для представления этих зависимостей.

Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области ("сущностей") и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

Семантические модели данных

Потребности проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области вызвали к жизни направление семантических моделей данных. При том, что любая развитая семантическая модель данных, как и реляционная модель, включает структурную, манипуляционную и целостную части, главным назначением семантических моделей является обеспечение возможности выражения семантики данных.

Прежде, чем мы коротко рассмотрим особенности одной из распространенных семантических моделей, остановимся на их возможных применениях.

Наиболее часто на практике семантическое моделирование используется на первой стадии проектирования базы данных. При этом в терминах семантической модели производится концептуальная схема базы данных, которая затем вручную преобразуется к реляционной (или какой-либо другой) схеме. Этот процесс выполняется под управлением методик, в которых достаточно четко оговорены все этапы такого преобразования.

Менее часто реализуется автоматизированная компиляция концептуальной схемы в реляционную. При этом известны два подхода: на основе явного представления концептуальной схемы как исходной информации для компилятора и построения интегрированных систем проектирования с автоматизированным созданием концептуальной схемы на основе интервью с экспертами предметной области. И в том, и в другом случае в результате производится реляционная схема базы данных в третьей нормальной форме (более точно следовало бы сказать, что автору неизвестны системы, обеспечивающие более высокий уровень нормализации).

Наконец, третья возможность, которая еще не вышла (или только выходит) за пределы исследовательских и экспериментальных проектов, - это работа с базой данных в семантической модели, т.е. СУБД, основанные на семантических моделях данных. При этом снова рассматриваются два варианта: обеспечение пользовательского интерфейса на основе семантической модели данных с автоматическим отображением конструкций в реляционную модель данных (это задача примерно такого же уровня сложности, как автоматическая компиляция концептуальной схемы базы данных в реляционную схему) и прямая реализация СУБД, основанная на какой-либо семантической модели данных. Наиболее близко ко второму подходу находятся современные объектно-ориентированные СУБД, модели данных которых по многим параметрам близки к семантическим моделям (хотя в некоторых аспектах они более мощны, а в некоторых - более слабы).

Модель Entity-Relationship (Сущность-Связи)

*****Основные понятия модели Entity-Relationship (Сущность-Связи)

Далее мы кратко рассмотрим некоторые черты одной из наиболее популярных семантических моделей данных - модель "Сущность-Связи" (часто ее называют кратко ER-моделью).

На использовании разновидностей ER-модели основано большинство современных подходов к проектированию баз данных (главным образом, реляционных). Модель была предложена Ченом (Chen) в 1976 г. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в системах CASE, поддерживающих автоматизированное проектирование реляционных баз данных. Среди множества разновидностей ER-моделей одна из наиболее развитых применяется в системе CASE фирмы

ORACLE. Ее мы и рассмотрим. Более точно, мы сосредоточимся на структурной части этой модели.

Основными понятиями ER-модели являются сущность, связь и атрибут.

Сущность - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности - это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных объектов этого типа.

Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности (это требование в некотором роде аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах).

Связь - это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

Связь представляется в виде линии, связывающей две сущности или ведущей от сущности к ней же самой. При это в месте "стыковки" связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много (many) экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный - прерывистой линией.

Как и сущность, связь - это типовое понятие, все экземпляры обеих пар связываемых сущностей подчиняются правилам связывания.

Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами.

Пример: Уникальным идентификатором сущности является атрибут, комбинация атрибутов, комбинация связей или комбинация связей и атрибутов, уникально отличающая любой экземпляр сущности от других экземпляров сущности того же типа.

*****Нормальные формы ER-схем

Как и в реляционных схемах баз данных, в ER-схемах вводится понятие нормальных форм, причем их смысл очень близко соответствует смыслу реляционных нормальных форм. Заметим, что формулировки нормальных форм ER-схем делают более понятным смысл нормализации реляционных схем. Мы приведем только очень краткие и неформальные определения трех первых нормальных форм.

В первой нормальной форме ER-схемы устраняются повторяющиеся атрибуты или группы атрибутов, т.е. производится выявление неявных сущностей, "замаскиро-ванных" под атрибуты.

Во второй нормальной форме устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность.

В третьей нормальной форме устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.

*****Более сложные элементы ER-модели

Мы остановились только на самых основных и наиболее очевидных понятиях ER-модели данных. К числу более сложных элементов модели относятся следующие:

Подтипы и супертипы сущностей. Как в языках программирования с развитыми типовыми системами (например, в языках объектно-ориентированного программирования), вводится возможность наследования типа сущности, исходя из одного или нескольких супертипов. Интересные нюансы связаны с необходимостью графического изображения этого механизма.

Связи "many-to-many". Иногда бывает необходимо связывать сущности таким образом, что с обоих концов связи могут присутствовать несколько экземпляров сущности (например, все члены кооператива сообща владеют имуществом кооператива). Для этого вводится разновидность связи "многие-со-многими".

Уточняемые степени связи. Иногда бывает полезно определить возможное количество экземпляров сущности, участвующих в данной связи (например, служащему разрешается участвовать не более, чем в трех проектах одновременно). Для выражения этого семантического ограничения разрешается указывать на конце связи ее максимальную или обязательную степень.

Каскадные удаления экземпляров сущностей. Некоторые связи бывают настолько сильными (конечно, в случае связи "один-ко-многим"), что при удалении опорного экземпляра сущности (соответствующего концу связи "один") нужно удалить и все экземпляры сущности, соответствующие концу связи "многие". Соответствующее требование "каскадного удаления" можно сформулировать при определении сущности.

Домены. Как и в случае реляционной модели данных бывает полезна возможность определения потенциально допустимого множества значений атрибута сущности (домена).

Эти и другие более сложные элементы модели данных "Сущность-Связи" делают ее существенно более мощной, но одновременно несколько усложняют ее использование. Конечно, при реальном использовании ER-диаграмм для проектирования баз данных необходимо ознакомиться со всеми возможностями.

Подробнее разберем только один из упомянутых элементов - подтип сущности.

Сущность может быть расщеплена на два или более взаимно исключающих подтипа, каждый из которых включает общие атрибуты и/или связи. Эти общие атрибуты и/или связи явно определяются один раз на более высоком уровне. В подтипах могут определяться собственные атрибуты и/или связи. В принципе подтипизация может продолжаться на более низких уровнях, но опыт показывает, что в большинстве случаев оказывается достаточно двух-трех уровней.

Сущность, на основе которой определяются подтипы, называется супертипом. Подтипы должны образовывать полное множество, т.е. любой экземпляр супертипа должен относиться к некоторому подтипу. Иногда для полноты приходится определять дополнительный подтип ПРОЧИЕ.

Иногда удобно иметь два или более разных разбиения сущности на подтипы. Например, сущность ЧЕЛОВЕК может быть разбита на подтипы по профессиональному признаку (ПРОГРАММИСТ, ДОЯРКА и т.д.), а может - по половому признаку (МУЖЧИНА, ЖЕНЩИНА).

*****Получение реляционной схемы из ER-схемы

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификатора, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

Шаг 4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

- все подтипы в одной таблице (а)
- для каждого подтипа - отдельная таблица (б)

При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется по крайней мере один столбец, содержащий код ТИПА; он становится частью первичного ключа.

При использовании метода (б) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

Шаг 7. Имеется два способа работы при наличии исключающих связей:

- общий домен (а)
- явные внешние ключи (б)

Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

Распределенные базы данных.

СУБД в архитектуре "клиент-сервер"

Применительно к системам баз данных архитектура "клиент-сервер" интересна и актуальна главным образом потому, что обеспечивает простое и относительно дешевое решение проблемы коллективного доступа к базам данных в локальной сети. В некотором роде системы баз данных, основанные на архитектуре "клиент-сервер", являются приближением к распределенным системам баз данных, конечно, существенно упрощенным приближением, но зато не требующим решения основного набора проблем действительно распределенных баз данных.

******* Основные принципы архитектуры «клиент-сервер»**

Система разбивается на две части, которые могут выполняться в разных узлах сети, - клиентскую и серверную части. Прикладная программа или конечный пользователь взаимодействуют с клиентской частью системы, которая в простейшем случае обеспечивает просто надсетевой интерфейс. Клиентская часть системы при потребности обращается по сети к серверной части. Заметим, что в развитых системах сетевое обращение к серверной части может и не понадобиться, если система может предугадывать потребности пользователя, и в клиентской части содержатся данные, способные удовлетворить его следующий запрос.

Интерфейс серверной части определен и фиксирован. Поэтому возможно создание новых клиентских частей существующей системы. Общим решением проблемы мобильности систем, основанных на архитектуре "клиент-сервер" является опора на программные пакеты, реализующие протоколы удаленного вызова процедур (RPC - Remote Procedure Call). При использовании таких средств обращение к сервису в удаленном узле выглядит как обычный вызов процедуры. Специфика сетевой среды и протоколов скрыта от прикладного программиста.

*******Серверы баз данных**

Термин "сервер баз данных" обычно используют для обозначения всей СУБД, основанной на архитектуре "клиент-сервер", включая и серверную, и клиентскую части. Такие системы предназначены для хранения и обеспечения доступа к базам данных.

Хотя обычно одна база данных целиком хранится в одном узле сети и поддерживается одним сервером, серверы баз данных представляют собой простое и дешевое приближение к

распределенным базам данных, поскольку общая база данных доступна для всех пользователей локальной сети.

***** **Принципы взаимодействия между клиентскими и серверными частями**

Доступ к базе данных от прикладной программы или пользователя производится путем обращения к клиентской части системы. В качестве основного интерфейса между клиентской и серверной частями выступает язык баз данных SQL.

Это язык по сути дела представляет собой текущий стандарт интерфейса СУБД в открытых системах. Собирательное название SQL-сервер относится ко всем серверам баз данных, основанных на SQL.

Серверы баз данных, интерфейс которых основан исключительно на языке SQL, обладают своими преимуществами и своими недостатками. Очевидное **преимущество** - стандартность интерфейса. В пределе, хотя пока это не совсем так, клиентские части любой SQL-ориентированной СУБД могли бы работать с любым SQL-сервером вне зависимости от того, кто его произвел.

Недостаток тоже довольно очевиден. При таком высоком уровне интерфейса между клиентской и серверной частями системы на стороне клиента работает слишком мало программ СУБД. Это нормально, если на стороне клиента используется маломощная рабочая станция. Но если клиентский компьютер обладает достаточной мощностью, то часто возникает желание возложить на него больше функций управления базами данных, разгрузив сервер, который является узким местом всей системы.

Одним из перспективных направлений СУБД является гибкое конфигурирование системы, при котором распределение функций между клиентской и пользовательской частями СУБД определяется при установке системы.

***** Преимущества протоколов удаленного вызова процедур
(RPC – Remote Procedre Call)

Упоминавшиеся выше протоколы удаленного вызова процедур особенно важны в системах управления базами данных, основанных на архитектуре "клиент-сервер".

Во-первых, использование механизма удаленных процедур позволяет действительно перераспределять функции между клиентской и серверной частями системы, поскольку в тексте программы удаленный вызов процедуры ничем не отличается от удаленного вызова, и следовательно, теоретически любой компонент системы может располагаться и на стороне сервера, и на стороне клиента.

Во-вторых, механизм удаленного вызова скрывает различия между взаимодействующими компьютерами. Физически неоднородная локальная сеть компьютеров приводится к логически однородной сети взаимодействующих программных компонентов. В результате пользователи не обязаны серьезно заботиться о разовой закупке совместимых серверов и рабочих станций.

*****Типичное разделение функций между клиентами и серверами

В типичном на сегодняшний день случае на стороне клиента СУБД работает только такое программное обеспечение, которое не имеет непосредственного доступа к базам данных, а обращается для этого к серверу с использованием языка SQL.

В некоторых случаях хотелось бы включить в состав клиентской части системы некоторые функции для работы с "локальным кэшем" базы данных, т.е. с той ее частью, которая интенсивно используется клиентской прикладной программой. В современной технологии это можно сделать только путем формального создания на стороне клиента локальной копии сервера базы данных и рассмотрения всей системы как набора взаимодействующих серверов.

С другой стороны, иногда хотелось бы перенести большую часть прикладной системы на сторону сервера, если разница в мощности клиентских рабочих станций и сервера чересчур велика. В общем-то при использовании RPC это сделать нетрудно. Но требуется, чтобы базовое программное обеспечение сервера действительно позволяло это.

*****Требования к аппаратным возможностям и базовому ПО клиентов и серверов

Из предыдущих рассуждений видно, что требования к аппаратуре и программному обеспечению клиентских и серверных компьютеров различаются в зависимости от вида использования системы.

Если разделение между клиентом и сервером достаточно жесткое (как в большинстве современных СУБД), то пользователям, работающим на рабочих станциях или персональных компьютерах, абсолютно все равно, какая аппаратура и операционная система работают на сервере, лишь бы он справлялся с возникающим потоком запросов.

Но если могут возникнуть потребности перераспределения функций между клиентом и сервером, то уже совсем не все равно, какие операционные системы используются.

Распределенные базы данных.

Основная задача систем управления распределенными базами данных состоит в обеспечении средства интеграции локальных баз данных, располагающихся в некоторых узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных как к единой базе данных.

При этом должны обеспечиваться:

- простота использования системы;
- возможности автономного ф-ирования при нарушениях связности сети или при административных потребностях;
- высокая степень эффективности.

******Разновидности распределенных систем**

Возможны **однородные и неоднородные распределенные базы данных**.

В однородном случае каждая локальная база данных управляется одной и той же СУБД.

В неоднородной системе локальные базы данных могут относиться даже к разным моделям данных. Сетевая интеграция неоднородных баз данных - это актуальная, но очень сложная проблема. Многие решения известны на теоретическом уровне, но пока не удается справиться с главной проблемой - недостаточной эффективностью интегрированных систем.

Заметим, что более успешно практически решается промежуточная задача - интеграция неоднородных SQL-ориентированных систем. Понятно, что этому в большой степени способствует стандартизация языка SQL и общее следование производителей СУБД принципам открытых систем.

*****Однородные распределенные СУБД**

Основную цель проектирования однородных распределенных СУБД можно сформулировать следующим образом: обеспечить средства интеграции локальных баз данных, располагающихся в узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных так, как если бы они были централизованы.

Для решения этих проблем было необходимо принять ряд проектных решений, касающихся декомпозиции исходного запроса, оптимального выбора способа выполнения запроса, согласованного выполнения транзакций, обеспечения синхронизации, обнаружения и разрешения распределенных тупиков, восстановления состояния баз данных после разного рода сбоев узлов сети.

Высокая степень эффективности системы является одним из наиболее ключевых требований к распределенным системам управления базами. Для достижения этой цели используются два основных приема.

Во-первых, выполнению запроса предшествует его компиляция. В ходе этого процесса производится поиск употребляемых в запросе имен объектов баз данных в распределенном каталоге и замена имен на внутренние идентификаторы; проверка прав доступа пользователя, от имени которого производится компиляция, на выполнение соответствующих операций над базами данных и выбор наиболее оптимального глобального плана выполнения запроса, который затем подвергается декомпозиции и по частям рассылается в соответствующие узлы сети, где производится выбор оптимальных локальных планов выполнения компонентов запроса и происходит генерация модулей доступа в машинных кодах. В результате множество действий производится на стадии компиляции до реального выполнения запроса. Обработанная посредством прекомпилятора прикладная программа, включающая предложения SQL, может в дальнейшем выполняться много раз без дополнительных накладных расходов.

Вторым средством повышения эффективности системы является возможность перемещения удаленных отношений в локальную базу данных.

Поддержание механизма транзакций - показатель уровня развитости СУБД. Корректное поддержание транзакций одновременно является основой обеспечения целостности баз данных (и поэтому транзакции вполне уместны и в однопользовательских персональных СУБД), а также составляют базис изолированности пользователей во многопользовательских системах.

Под транзакцией понимается неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации) такая, что либо результаты всех операторов, входящих в транзакцию, отображаются в БД, либо воздействие всех этих операторов полностью отсутствует. Лозунг транзакции - "Все или ничего": при завершении транзакции оператором COMMIT результаты гарантированно фиксируются во внешней памяти (смысл слова commit - "зафиксировать" результаты транзакции); при завершении транзакции оператором ROLLBACK результаты гарантированно отсутствуют во внешней памяти (смысл слова rollback - ликвидировать результаты транзакции).

Транзакции и целостность баз данных

Понятие транзакции имеет непосредственную связь с понятием целостности БД. Очень часто БД может обладать такими ограничениями целостности, которые просто невозможно не нарушить, выполняя только один оператор изменения БД.

Поэтому для поддержания подобных ограничений целостности допускается их нарушение внутри транзакции с тем условием, чтобы к моменту завершения транзакции условия целостности были соблюдены. В системах с развитыми средствами ограничения и контроля целостности каждая транзакция начинается при целостном состоянии БД и должна оставить это состояние целостными после своего завершения. Несоблюдение этого условия приводит к тому, что вместо фиксации результатов транзакции происходит ее откат (т.е. вместо оператора COMMIT выполняется оператор ROLLBACK), и БД остается в таком состоянии, в котором находилась к моменту начала транзакции, т.е. в целостном состоянии.

Если быть немного более точным, различаются два вида ограничений целостности: немедленно проверяемые и откладываемые. К немедленно проверяемым ограничениям целостности относятся такие ограничения, проверку которых бессмысленно или даже невозможно откладывать. Примером ограничения, проверку которого откладывать бессмысленно, являются ограничения домена (возраст сотрудника не может превышать 150 лет). Более сложным ограничением, проверку которого невозможно отложить, является следующее: зарплата сотрудника не может быть увеличена за одну операцию более, чем на 100,000 рублей. Немедленно проверяемые ограничения целостности соответствуют уровню отдельных операторов языкового уровня СУБД. При их нарушениях не производится откат транзакции, а лишь отвергается соответствующий оператор.

Откладываемые ограничения целостности - это ограничения на базу данных, а не на какие-либо отдельные операции. По умолчанию такие ограничения проверяются при конце транзакции, и их нарушение вызывает автоматическую замену оператора COMMIT на оператор ROLLBACK. Однако в некоторых системах поддерживается специальный оператор насильственной проверки ограничений целостности внутри транзакции. Если после выполнения такого оператора обнаруживается, что условия целостности не выполнены, пользователь может сам выполнить оператор ROLLBACK или постараться устранить причины нецелостного состояния базы данных внутри транзакции (видимо, это осмысленно только при использовании интерактивного режима работы).

И еще одно замечание. С точки зрения внешнего представления в момент завершения транзакции проверяются все откладываемые ограничения целостности, определенные в этой базе данных. Однако при реализации стремятся при выполнении транзакции динамически выделить те ограничения целостности, которые действительно могли бы быть нарушены. Например, если при выполнении транзакции над базой данных СОТРУДНИКИ-ОТДЕЛЫ в ней не выполнялись операторы вставки или удаления кортежей из отношения СОТРУДНИКИ, то проверять упоминавшееся выше ограничение целостности не требуется (а проверка подобных ограничений вызывает достаточно большую работу).

Изолированность пользователей

Во многопользовательских системах с одной базой данных одновременно могут работать несколько пользователей или прикладных программ. Предельной задачей системы является

обеспечение изолированности пользователей, т.е. создание достоверной и надежной иллюзии того, что каждый из пользователей работает с БД в одиночку.

В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей. Действительно, если с каждым сеансом работы с базой данных ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием базы данных, т.е. с таким состоянием, в котором база данных могла бы находиться, даже если бы пользователь работал с ней в одиночку.

При соблюдении обязательного требования поддержания целостности базы данных возможны следующие **уровни изолированности транзакций:**

Первый уровень - отсутствие потерянных изменений.

// Рассмотрим следующий сценарий совместного выполнения двух транзакций. Транзакция 1 изменяет объект базы данных A. До завершения транзакции 1 транзакция 2 также изменяет объект A. Транзакция 2 завершается оператором ROLLBACK (например, по причине нарушения ограничений целостности). Тогда при повторном чтении объекта A транзакция 1 не видит изменений этого объекта, произведенных ранее. Такая ситуация называется ситуацией потерянных изменений. Естественно, она противоречит требованию изолированности пользователей. Чтобы избежать такой ситуации в транзакции 1 требуется, чтобы до завершения транзакции 1 никакая другая транзакция не могла изменять объект A. Отсутствие потерянных изменений является минимальным требованием к СУБД по части синхронизации параллельно выполняемых транзакций.

Второй уровень - отсутствие чтения "грязных данных".

// Рассмотрим следующий сценарий совместного выполнения транзакций 1 и 2. Транзакция 1 изменяет объект базы данных A. Параллельно с этим транзакция 2 читает объект A. Поскольку операция изменения еще не завершена, транзакция 2 видит несогласованные "грязные" данные (в частности, операция транзакции 1 может быть отвернута при проверке немедленно проверяемого ограничения целостности). Это тоже не соответствует требованию изолированности пользователей (каждый пользователь начинает свою транзакцию при согласованном состоянии базы данных и в праве ожидать видеть согласованные данные). Чтобы избежать ситуации чтения "грязных" данных, до завершения транзакции 1, изменившей объект A, никакая другая транзакция не должна читать объект A (минимальным требованием является блокировка чтения объекта A до завершения операции его изменения в транзакции 1).

Третий уровень - отсутствие неповторяющихся чтений.

// Рассмотрим следующий сценарий. Транзакция 1 читает объект базы данных A. До завершения транзакции 1 транзакция 2 изменяет объект A и успешно завершается оператором COMMIT. Транзакция 1 повторно читает объект A и видит его измененное состояние. Чтобы избежать неповторяющихся чтений, до завершения транзакции 1 никакая другая транзакция не должна изменять объект A. В большинстве систем это является максимальным требованием к синхронизации транзакций, хотя, как мы увидим немного позже, отсутствие неповторяющихся чтений еще не гарантирует реальной изолированности пользователей.

Заметим, что существует возможность обеспечения разных уровней изолированности для разных транзакций, выполняющихся в одной системе баз.

Существует ряд приложений, для которых первого уровня достаточно (например, прикладные или системные статистические утилиты, для которых некорректность индивидуальных данных несущественна). При этом удается существенно сократить накладные расходы СУБД и повысить общую эффективность.

К более тонким проблемам изолированности транзакций относится так называемая проблема кортежей-"фантомов", вызывающая ситуации, которые также противоречат изолированности пользователей.

Чтобы избежать появления кортежей-фантомов, требуется более высокий "логический" уровень синхронизации транзакций. Идеи такой синхронизации (предикатные синхронизационные захваты) известны давно, но в большинстве систем не реализованы.

// Рассмотрим следующий сценарий. Транзакция 1 выполняет оператор A выборки кортежей отношения R с условием выборки S (т.е. выбирается часть кортежей отношения R, удовлетворяющих условию S). До завершения транзакции 1 транзакция 2 вставляет в отношение R новый кортеж r, удовлетворяющий условию S, и успешно завершается. Транзакция 1 повторно выполняет оператор A, и в результате появляется кортеж, который отсутствовал при первом выполнении оператора. Конечно, такая ситуация противоречит идее

изолированности транзакций и может возникнуть даже на третьем уровне изолированности транзакций.

*******Сериализация транзакций**

Понятно, что для того, чтобы добиться изолированности транзакций, в СУБД должны использоваться какие-либо методы регулирования совместного выполнения транзакций.

План (способ) выполнения набора транзакций называется сериальным, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций.

Сериализация транзакций - это механизм их выполнения по некоторому сериальному плану. Обеспечение такого механизма является основной функцией компонента СУБД, ответственного за управление транзакциями. Система, в которой поддерживается сериализация транзакций обеспечивает реальную изолированность пользователей.

Основная реализационная проблема состоит в выборе метода сериализации набора транзакций, который не слишком ограничивал бы их параллельность. Приходящим на ум тривиальным решением является действительно последовательное выполнение транзакций. Но существуют ситуации, в которых можно выполнять операторы разных транзакций в любом порядке с сохранением сериальности. Примерами могут служить только читающие транзакции, а также транзакции, не конфликтующие по объектам базы данных.

- Между транзакциями могут существовать следующие виды конфликтов:
- W-W - транзакция 2 пытается изменять объект, измененный не закончившейся транзакцией 1;
- R-W - транзакция 2 пытается изменять объект, прочитанный не закончившейся транзакцией 1;
- W-R - транзакция 2 пытается читать объект, измененный не закончившейся транзакцией 1.

Практические методы сериализации транзакций основывается на учете этих конфликтов.

*******Интегрированные или федеративные системы и мультибазы данных**

Направление интегрированных или федеративных систем неоднородных БД и мульти-БД появилось в связи с необходимостью комплексирования систем БД, основанных на разных моделях данных и управляемых разными СУБД.

Основной задачей интеграции неоднородных БД является предоставление пользователям интегрированной системы глобальной схемы БД, представленной в некоторой модели данных, и автоматическое преобразование операторов манипулирования БД глобального уровня в операторы, понятные соответствующим локальным СУБД. В теоретическом плане проблемы преобразования решены, имеются реализации.

При строгой интеграции неоднородных БД локальные системы БД утрачивают свою автономность. После включения локальной БД в федеративную систему все дальнейшие действия с ней, включая администрирование, должны вестись на глобальном уровне. Поскольку пользователи часто не соглашаются утрачивать локальную автономность, желая тем не менее иметь возможность работать со всеми локальными СУБД на одном языке и формулировать запросы с одновременным указанием разных локальных БД, развивается направление мульти-БД. В системах мульти-БД не поддерживается глобальная схема интегрированной БД и применяются специальные способы именования для доступа к объектам локальных БД. Как правило, в таких системах на глобальном уровне допускается только выборка данных. Это позволяет сохранить автономность локальных БД.

Как правило, интегрировать приходится неоднородные БД, распределенные в вычислительной сети. Это в значительной степени усложняет реализацию. Дополнительно к собственным проблемам интеграции приходится решать все проблемы, присущие распределенным СУБД: управление глобальными транзакциями, сетевую оптимизацию запросов и т.д. Очень трудно добиться эффективности.

Как правило, для внешнего представления интегрированных и мульти-БД используется (иногда расширенная) реляционная модель данных. В последнее время все чаще предлагается использовать объектно-ориентированные модели, но на практике пока основой является реляционная модель. Поэтому, в частности, включение в интегрированную систему локальной

реляционной СУБД существенно проще и эффективнее, чем включение СУБД, основанной на другой модели данных.

Реляционный язык SQL.

Язык для взаимодействия с БД SQL появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (Structered English Query Language) только частично отражает суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле уже являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; возможности определения структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов. В языке отсутствовали средства синхронизации доступа к объектам БД со стороны параллельно выполняемых транзакций: с самого начала предполагалось, что необходимую синхронизацию неявно выполняет СУБД.

SQL представляет собой некоторую комбинацию реляционного исчисления кортежей и реляционной алгебры, причем до сих пор нет общего согласия, к какому из классических языков он ближе. При этом возможности SQL шире, чем у этих базовых реляционных языков.

В настоящее время SQL реализован практически во всех коммерческих реляиционных СУБД, все фирмы провозглашают соответствие своей реализации стандарту SQL, и на самом деле реализованные диалекты SQL очень близки.

Рассмотрим **основные особенности стандарта языка SQL 1989**г

***** Типы данных SQL

В языке SQL/89 поддерживаются следующие типы данных: CHARACTER, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION. Эти типы данных классифицируются на типы строк символов, точных чисел и приблизительных чисел.

К первому классу относится CHARACTER. Спецификатор типа имеет вид CHARACTER (lenght), где lenght задает длину строк данного типа. Заметим, что в SQL/89 нет типа строк переменного размера, хотя во многих реализациях они допускаются. Литеральные строки символов изображаются в виде 'последовательность символов' (например, 'example').

Представителями второго класса типов являются NUMERIC, DECIMAL (или DEC), INTEGER (или INT) и SMALLINT. Спецификатор типа NUMERIC имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представляемые с точностью precision и масштабом scale.

Спецификатор типа DECIMAL (или DEC) имеет вид DECIMAL [(precision [, scale]).

Наконец, в классу типов данных приблизительных чисел относятся типы FLOAT, REAL и DOUBLE PRECISION. Спецификатор типа FLOAT имеет вид FLOAT [(precision)].

Большинстве реализаций SQL поддерживаются некоторые дополнительные типы данных, например, DATE, TIME, INTERVAL, MONEY.

*****Определение таблицы

Оператор определения таблицы имеет следующий синтаксис:

<table definition> ::=
  CREATE TABLE <table name> (<table element>
  [{,<table element>}...])
<table element> ::=
  <column definition>

| <table constraint definition>

Кроме имени таблицы, в операторе специфицируется список элементов таблицы, каждый из которых служит либо для определения столбца, либо для определения ограничения целостности определяемой таблицы. Требуется наличие хотя бы одного определения столбца. Оператор CREATE TABLE определяет так называемую базовую таблицу, т.е. реальное хранилище данных.

Для определения столбцов таблицы и ограничений целостности используются специальные операторы, которые должны быть вложены в оператор определения таблицы.

***** Определение столбца

Оператор определения столбца описывается следующими синтаксическими правилами:

<column definition> ::=
  <column name> <data type>
  [<default clause>] [<column constraint>...]
<default clause> ::=
  DEFAULT { <literal> | USER | NULL }
<column constraint> ::=
  NOT NULL [<unique specification>]
 | <references specification>
 | CHECK (<search condition>)

Как видно, кроме обязательной части, в которой определяется имя столбца и его тип данных, определение столбца может содержать два необязательных раздела: раздел значения столбца по умолчанию и раздел ограничений целостности столбца.

В разделе значения по умолчанию указывается значение, которое должно быть помещено с строку, заносимую в данную таблицу, если значение данного столбца явно не указано. Значение по умолчанию может быть указано в виде литеральной константы с типом, соответствующим типу столбца; путем задания ключевого слова USER, которому при выполнении оператора занесения строки соответствует символьная строка, содержащая имя текущего пользователя (в этом случае столбец должен иметь тип символьных строк); или путем задания ключевого слова NULL, означающего, что значением по умолчанию является неопределенное значение. Если значение столбца по умолчанию не специфицировано, и в разделе ограничений целостности столбца указано NOT NULL, то попытка занести в таблицу строку с неспецифицированным значением данного столбца приведет к ошибке.

Указание в разделе ограничений целостности NOT NULL приводит к неявному порождению проверочного ограничения целостности для всей таблицы (см. следующий подраздел) "CHECK (C IS NOT NULL)" (где C - имя данного столбца). Если ограничение NOT NULL не указано, и раздел умолчаний отсутствует, то неявно порождается раздел умолчаний DEFAULT NULL. Если указана спецификация уникальности, то порождается соответствующая спецификация уникальности для таблицы.

Если в разделе ограничений целостности указано ограничение по ссылкам данного столбца (<reference specification>), то порождается соответствующее определение ограничения по ссылкам для таблицы:

FOREIGN KEY(C) <reference specification>.

Наконец, если указано проверочное ограничение столбца, то условие поиска этого ограничения должно ссылаться только на данный столбец, и неявно порождается соответствующее проверочное ограничение для всей таблицы.

***** Определение ограничений целостности таблицы

Ограничения целостности таблицы обладают следующим синтаксисом:

<table constraint definition> ::=
  <unique constraint definition>
 | <referential constraint definition>
 | <check constraint definition>
<unique constraint definition> ::=
  <unique specification> (<unique column list>)
<unique specification> ::= UNIQUE | PRIMARY KEY
<unique column list> ::= <column name> [{,<column name>}...]
<referential constraint definition> ::=
  FOREIGN KEY (<referencing columns>) <references specification>

<references specification> ::=
REFERENCES <referenced table and columns>
<referencing columns> ::= <reference column list>
<referenced table and columns> ::=
<table name> [(<reference column list>)]
<reference column list> ::= <column name> [{,<column name>}...]
<check constraint definition> ::= CHECK (<search condition>)

Для одной таблицы может быть задано несколько ограничений целостности, в том числе те, которые неявно порождаются ограничениями целостности столбцов. Стандарт SQL/89 устанавливает, что ограничения таблицы фактически проверяются при выполнении каждого оператора SQL.

Замечание: Наличие правильно подобранного набора ограничений БД очень важно для надежного функционирования прикладной информационной системы. Вместе с тем, в некоторых СУБД ограничения целостности практически не поддерживаются. Поэтому при проектировании прикладной системы необходимо принять решение о том, что более существенно: рассчитывать на поддержку ограничений целостности, но ограничить набор возможных СУБД, или отказаться от их использования на уровне SQL, сохранив возможность использования не самых современных СУБД.

*****Определение представлений

Механизм представлений (view) является мощным средством языка SQL, позволяющим скрыть реальную структуру БД от некоторых пользователей за счет определения представления БД, которое реально является некоторым хранимым в БД запросом с именованными столбцами, а для пользователя ничем не отличается от базовой таблицы БД (с учетом технических ограничений). Любая реализация должна гарантировать, что состояние представляемой таблицы точно соответствует состоянию базовых таблиц, на которых определено представление. Обычно вычисление представляемой таблицы (материализация соответствующего запроса) производится каждый раз при использовании представления.

В стандарте SQL/89 оператор определения представления имеет следующий синтаксис:

<view definition> ::=
CREATE VIEW <table name> [(<view column list>)]
AS <query specification> [WITH CHECK OPTION]
<view column list> ::= <column name> [{,<column name>}...]

Определяемая представляемая таблица V является изменяемой (т.е. по отношению к V можно использовать операторы DELETE и UPDATE) в том и только в том случае, если выполняются следующие условия для спецификации запроса:

В списке выборки не указано ключевое слово DISTINCT;

Каждое арифметическое выражение в списке выборки представляет собой одну спецификацию столбца, и спецификация одного столбца не появляется более одного раза;

В разделе FROM указана только одна таблица, являющаяся либо базовой таблицей, либо изменяемой представляемой таблицей;

В условии выборки раздела WHERE не используются подзапросы;

В табличном выражении отсутствуют разделы GROUP BY и HAVING.

Замечание: Эти ограничения являются очень сильными. В реализациях они могут быть ослаблены. Но если стремиться к мобильности, не следует пользоваться расширенными возможностями.

Если в списке выборки спецификации запроса имеется хотя бы одно арифметическое выражение, состоящее не из одной спецификации столбца, или если одно имя столбца участвует в списке выборки более одного раза, определение представления должно содержать список имен столбцов представляемой таблицы. Более просто, нужно явно именовать столбцы представляемой таблицы, если эти имена не наследуются от столбцов таблиц раздела FROM спецификации запроса.

Требование WITH CHECK OPTION в определении представления имеет смысл только в случае определения изменяемой представляемой таблицы, которая определяется спецификацией запроса, содержащей раздел WHERE. При наличии этого требования не допускаются изменения представляемой таблицы, которые приводят к появлению в базовых

таблиц строк, не видимых в представляемой таблице (т.е. таких строк, которые не удовлетворяют условию поиска раздела WHERE спецификации запроса). Если WITH CHECK OPTION в определении представления отсутствует, такой контроль не производится.

***** Определение привилегий

В соответствии с идеологией языка SQL контроль прав доступа данного пользователя к таблицам БД производится на основе механизма привилегий. Фактически, этот механизм состоит в том, что для выполнения любого действия над таблицей пользователь должен обладать соответствующей привилегией (реально все возможные действия описываются фиксированным стандартным набором привилегий). Пользователь, создавший таблицу, автоматически становится владельцем всех возможных привилегий на выполнение операций над этой таблицей. В число этих привилегий входит привилегия на передачу всех или некоторых привилегий по отношению к данной таблице другому пользователю, включая привилегию на передачу привилегий. Иногда поддерживается и обратная операция изъятия привилегий от пользователя, ранее их получившего.

В SQL/89 определяется упрощенная схема механизма привилегий. Во-первых, "раздача" привилегий возможна только при определении таблицы. Во-вторых, пользователь, получивший некоторые привилегии от других пользователей, может передать их дальше только при определении схемы.

Определение привилегий производится в следующем синтаксисе:

<privilege definition> ::=
  GRANT <privileges> ON <table name> TO <grantee>
  [{,<grantee>}...] [WITH GRANT OPTION]
<privileges> ::=
  ALL PRIVILEGES
 | <action> [{,<action>}...]
<action> ::= SELECT | INSERT | DELETE
        | UPDATE [(<grant column list>)]
        | REFERENCES [(<grant column list>]
<grant column list> ::= <column name> [{,<column name>}...]
<grantee> ::= PUBLIC | <authorization identifier>

Привилегией REFERENCES по отношению к указанным столбцам таблицы T1 необходимо обладать, чтобы иметь возможность при определении таблицы T определить ограничение по ссылкам между этой таблицей и существующей к этому моменту таблицей T1.

*****Структура запросов

Для того, чтобы можно было более или менее точно рассказать про структуру запросов в стандарте SQL/89, необходимо начать со сводки синтаксических правил:

<cursor specification> ::=
  <query expression> [<order by clause>]
<query expression> ::=
  <query term>
| <query expression> UNION [ALL] <query term>
<query term> ::=
  <query specification>
| (<query expression>)
<query specification> ::=
  (SELECT [ALL | DISTINCT] <select list> <table expression>)
<select statement> ::=
  SELECT [ALL | DISTINCT] <select list>
  INTO <select target list> <table expression>
<subquery> ::=
  (SELECT [ALL | DISTINCT] <result specification>
  <table expression>
<table expression> ::=

<from clause>
[<where clause>]
[<group by clause>]
[<having clause>]

Язык допускает три типа синтаксических конструкций, начинающихся с ключевого слова SELECT: спецификация курсора (cursor specification), оператор выборки (select statement) и подзапрос (subquery). Основой всех них является синтаксическая конструкция "табличное выражение (table expression)". Семантика табличного выражения состоит в том, что на основе последовательного применения разделов from, where, group by и having из заданных в разделе from таблиц строится некоторая новая результирующая таблица, порядок следования строк которой не определен и среди строк которой могут находиться дубликаты (т.е. в общем случае таблица-результат табличного выражения является мультимножеством строк). На самом деле именно структура табличного выражения наибольшим образом характеризует структуру запросов языка SQL/89. Мы рассмотрим ниже структуру и смысл разделов табличного выражения ниже, но до этого немного подробнее обсудим три упомянутые конструкции, включающие табличные выражения.

Спецификация курсора

Наиболее общей является конструкция "спецификация курсора". Курсор - это понятие языка SQL, позволяющее с помощью набора специальных операторов получить построчный доступ к результату запроса к БД. К табличным выражениям, участвующим в спецификации курсора, не предъявляются какие-либо ограничения. Как видно из сводки синтаксических правил, при определении спецификации курсора используются три дополнительных конструкции: спецификация запроса, выражение запросов и раздел ORDER BY.

Спецификация запроса

В спецификации запроса задается список выборки (список арифметических выражений над значениями столбцов результата табличного выражения и констант). В результате применения списка выборки к результату табличного выражения производится построение новой таблицы, содержащей то же число строк, но вообще говоря другое число столбцов, содержащих результаты вычисления соответствующих арифметических выражений из списка выборки. Кроме того, в спецификации запроса могут содержаться ключевые слова ALL или DISTINCT. При наличии ключевого слова DISTINCT из таблицы, полученной применением списка выборки к результату табличного выражения, удаляются строки-дубликаты; при указании ALL (или просто при отсутствии DISTINCT) удаление строк-дубликатов не производится.

Выражение запросов

Выражение запросов - это выражение, строящееся по указанным синтаксическим правилам на основе спецификаций запросов. Единственной операцией, которую разрешается использовать в выражениях запросов, является операция UNION (объединение таблиц) с возможной разновидностью UNION ALL. К таблицам-операндам выражения запросов предъявляется то требование, что все они должны содержать одно и то же число столбцов, и соответствующие столбцы всех операндов должны быть одного и того же типа. Выражение запросов вычисляется слева направо с учетом скобок. При выполнении операции UNION производится обычное теоретико-множественное объединение операндов, т.е. из результирующей таблицы удаляются дубликаты. При выполнении операции UNION ALL образуется результирующая таблица, в которой могут содержаться строки-дубликаты.

Раздел ORDER BY

Наконец, раздел ORDER BY позволяет установить желаемый порядок просмотра результата выражения запросов. Синтаксис ORDER BY следующий:

<order by clause> ::=
ORDER BY <sort specification>
[{,<sort specification>}...]
<sort specification> ::=
{<unsigned integer> | <column specification>}
[ASC | DESC]

Как видно из этих синтаксических правил, фактически задается список столбцов результата выражения запросов, и для каждого столбца указывается порядок просмотра строк результата в зависимости от значений этого столбца (ASC - по возрастанию (умолчание), DESC - по убыванию). Столбцы можно задавать их именами в том и только в том случае, когда (1)

выражение запросов не содержит операций UNION или UNION ALL и (2) в списке выборки спецификации запроса этому столбцу соответствует арифметическое выражение, состоящее только из имени столбца. Во всех остальных случаях в разделе ORDER BY должен указываться порядковый номер столбца в таблице-результате выражения запросов.

***** Оператор выборки

Оператор выборки - это отдельный оператор языка SQL/89, позволяющий получить результат запроса в прикладной программе без привлечения курсора. Поэтому оператор выборки имеет синтаксис, отличающийся от синтаксиса спецификации курсора, и при его выполнении возникают ограничения на результат табличного выражения. Фактически, и то, и другое диктуется спецификой оператора выборки как одиночного оператора SQL: при его выполнении результат должен быть помещен в переменные прикладной программы. Поэтому в операторе появляется раздел INTO, содержащий список переменных прикладной программы, и возникает то ограничение, что результирующая таблица должна содержать не более одной строки. Соответственно, результат базового табличного выражения должен содержать не более одной строки, если оператор выборки не содержит спецификации DISTINCT, и таблица, полученная применением списка выборки к результату табличного выражения, не должна содержать более одной несовпадающих строк, если спецификация DISTINCT задана.

Замечание: В диалекте SQL СУБД Oracle поддерживается расширенный вариант оператора выборки, результатом которого не обязательно является таблица из одной строки. Такое расширение не поддерживается ни в SQL/89, ни в SQL/92.

***** Подзапрос

Наконец, последняя конструкция SQL/89, которая может содержать табличные выражения, - это подзапрос, т.е. запрос, который может входить в предикат условия выборки оператора SQL. В SQL/89 к подзапросам применяется то ограничение, что результирующая таблица должна содержать в точности один столбец. Поэтому в синтаксических правилах, определяющих подзапрос, вместо списка выборки указано "выражение, вычисляющее значение", т.е. арифметическое выражение. Заметим еще, что поскольку подзапрос всегда вложен в некоторый другой оператор SQL, то в качестве констант в арифметическом выражении выборки и логических выражениях разделов WHERE и HAVING можно использовать значения столбцов текущих строк таблиц, участвующих в (под)запросах более внешнего уровня. Более подробно об этом см. ниже, при описании семантики табличных выражений.

*****Табличное выражение

Стандарт SQL/89 рекомендует рассматривать вычисление табличного выражения как последовательное применение разделов FROM, WHERE, GROUP BY и HAVING к таблицам, заданным в списке FROM. Раздел FROM имеет следующий синтаксис:

<from clause> ::=
  FROM <table reference>
 ({,<table reference>}...]
<table reference> ::=
  <table name> [<correlation name>]

Раздел FROM

Результатом выполнения раздела FROM является расширенное декартово произведение таблиц, заданных списком таблиц раздела FROM. Расширенное декартово произведение (расширенное, потому что в качестве операндов и результата допускаются мультимножества) в стандарте определяется следующим образом:

"Расширенное произведение R есть мультимножество всех строк r таких, что r является конкатенацией строк из всех идентифицированных таблиц в том порядке, в котором они идентифицированы. Мощность R есть произведение мощностей идентифицированных таблиц. Порядковый номер столбца в R есть n+s, где n - порядковый номер порождающего столбца в именованной таблице T, а s - сумма степеней всех таблиц, идентифицированных до T в разделе FROM".

Как видно из синтаксиса, рядом с именем таблицы можно указывать еще одно имя "correlation name". Фактически, это некоторый синоним имени таблицы, который можно использовать в других разделах табличного выражения для ссылки на строки именно этого вхождения таблицы.

Если табличное выражение содержит только раздел FROM (это единственный обязательный раздел табличного выражения), то результат табличного выражения совпадает с результатом раздела FROM.

Раздел WHERE

Если в табличном выражении присутствует раздел WHERE, то следующим вычисляется он. Синтаксис раздела WHERE следующий:

<where clause> ::= WHERE <search condition>
<search condition> ::=
  <boolean term>
 (<search condition> OR <boolean term>
<Boolean term> ::=
  <boolean factor>
 (<boolean term> AND <boolean factor>
<boolean factor> ::= [NOT] <boolean primary>
<boolean primary> ::= <predicate> | (<search condition>)

Вычисление раздела WHERE производится по следующим правилам: Пусть R - результат вычисления раздела FROM. Тогда условие поиска применяется ко всем строкам R, и результатом раздела WHERE является таблица, состоящая из тех строк R, для которого результатом вычисления условия поиска является true. Если условие выборки включает подзапросы, то каждый подзапрос вычисляется для каждого кортежа таблицы R (в стандарте используется термин "effectively" в том смысле, что результат должен быть таким, как если бы каждый подзапрос действительно вычислялся заново для каждого кортежа R).

Заметим, что поскольку SQL/89 допускает наличие в базе данных неопределенных значений, то вычисление условия поиска производится не в булевой, а в трехзначной логике со значениями true, false и unknown (неизвестно). Для любого предиката известно, в каких ситуациях он может порождать значение unknown. Булевские операции AND, OR и NOT работают в трехзначной логике следующим образом:

true AND unknown = unknown
unknown AND true = unknown
unknown AND unknown = unknown
true OR unknown = true
unknown OR true = true
unknown OR unknown = unknown
NOT unknown = unknown

Среди предикатов условия поиска в соответствии с SQL/89 могут находиться следующие предикаты: предикат сравнения, предикат between, предикат in, предикат like, предикат null, предикат с квантором и предикат exists. Сразу заметим, что во всех реализациях SQL на эффективность выполнения запроса существенно влияет наличие в условии поиска простых предикатов сравнения (предикатов, задающих сравнение столбца таблицы с константой). Наличие таких предикатов позволяет СУБД использовать индексы при выполнении запроса, т.е. избегать полного просмотра таблицы. Хотя в принципе язык SQL позволяет пользователям не заботиться о конкретном наборе предикатов в условии выборки (лишь бы они были синтаксически и семантически правильны), при реальном использовании SQL-ориентированных СУБД такие технические детали стоит иметь в виду.

Предикат сравнения

Синтаксис предиката сравнения определяется следующими правилами:

<comparison predicate> ::=
  <value expression> <comp op>
 {<value expression> | <subquery>}
<comp op> ::=
  = | <> | < | > | <= | >=

Через "<>" обозначается операция "неравенства". Арифметические выражения левой и правой частей предиката сравнения строятся по общим правилам построения арифметических

выражений и могут включать в общем случае имена столбцов таблиц из раздела FROM и константы. Типы данных арифметических выражений должны быть сравнимыми (например, если тип столбца a таблицы A является типом символьных строк, то предикат "a = 5" недопустим).

Если правый операнд операции сравнения задается подзапросом, то дополнительным ограничением является то, что мощность результата подзапроса должна быть не более единицы. Если хотя бы один из операндов операции сравнения имеет неопределенное значение, или если правый операнд является подзапросом с пустым результатом, то значение предиката сравнения равно unknown.

Заметим, что значение арифметического выражения не определено, если в его вычислении участвует хотя бы одно неопределенное значение. Еще одно важное замечание из стандарта SQL/89: в контексте GROUP BY, DISTINCT и ORDER BY неопределенное значение выступает как специальный вид определенного значения, т.е. возможно, например, образование группы строк, значение указанного столбца которых является неопределенным. Для обеспечения переносимости прикладных программ нужно внимательно оценивать специфику работы с неопределенными значениями в конкретной СУБД.

Предикат between

Предикат between имеет следующий синтаксис:

<between predicate> ::=
  <value expression>
  [NOT] BETWEEN <value expression> AND <value expression>

Результат "x BETWEEN y AND z" тот же самый, что результат "x >= y AND x <= z". Результат "x NOT BETWEEN y AND z" тот же самый, что результат "NOT (x BETWEEN y AND z)".

Предикат in

Предикат in определяется следующими синтаксическими правилами:

<in predicate> ::=
  <value expression> [NOT] IN
 {<subquery> | (<in value list>)}
<in value list> ::=
  <value specification>
 {,<value specification>}...

Типы левого операнда и значений из списка правого операнда (напомним, что результирующая таблица подзапроса должна содержать ровно один столбец) должны быть сравнимыми.

Значение предиката равно true в том и только в том случае, когда значение левого операнда совпадает хотя бы с одним значением списка правого операнда. Если список правого операнда пуст (так может быть, если правый операнд задается подзапросом), или значение "подразумеваемого" предиката сравнения x = y (где x - значение арифметического выражения левого операнда) равно false для каждого элемента y списка правого операнда, то значение предиката in равно false. В противном случае значение предиката in равно unknown. По определению значение предиката "x NOT IN S" равно значению предиката "NOT (x IN S)".

Предикат like

Предикат like имеет следующий синтаксис:

<like predicate> ::=
  <column specification> [NOT] LIKE <pattern>
  [ESCAPE <escape character>]
<pattern> ::= <value specification>
  <escape character> ::= <value specification>

Типы данных столбца левого операнда и образца должны быть типами символьных строк. В разделе ESCAPE должен специфицироваться одиночный символ.

Значение предиката равно true, если pattern является подстрокой заданного столбца. При этом, если раздел ESCAPE отсутствует, то при сопоставлении шаблона со строкой производится специальная интерпретация двух символов шаблона: символ подчеркивания ("_") обозначает любой одиночный символ; символ процента ("%") обозначает последовательность произвольных символов произвольной длины (может быть, нулевой).

Если же раздел ESCAPE присутствует и специфицирует некоторый одиночный символ x, то пары символов "x_" и "x%" представляют одиночные символы "_" и "%" соответственно.

Значение предиката like есть unknown, если значение столбца, либо шаблона не определено.

Значение предиката "x NOT LIKE y ESCAPE z" совпадает со значением "NOT x LIKE y ESCAPE z".

Предикат null

Предикат null описывается синтаксическим правилом:

<null predicate> ::=
  <column specification> IS [NOT] NULL

Этот предикат всегда принимает значения true или false. При этом значение "x IS NULL" равно true тогда и только тогда, когда значение x не определено. Значение предиката "x NOT IS NULL" равно значению "NOT x IS NULL".

Предикат с квантором

Предикат с квантором имеет следующий синтаксис:

<quantified predicate> ::=
  <value expression> <comp op> <quantifier> <subquery>
<quantifier> ::=
  <all> | <some>
<all> ::= ALL
<some> ::= SOME | ANY

Обозначим через x результат вычисления арифметического выражения левой части предиката, а через S результат вычисления подзапроса.

Предикат "x <comp op> ALL S" имеет значение true, если S пусто или значение предиката "x <comp op> s" равно true для каждого s, входящего в S. Предикат "x <comp op> ALL S" имеет значение false, если значение предиката "x <comp op> s" равно false хотя бы для одного s, входящего в S. В остальных случаях значение предиката "x <comp op> ALL S" равно unknown.

Предикат "x <comp op> SOME S" имеет значение false, если S пусто или значение предиката "x <comp op> s" равно false для каждого s, входящего в S. Предикат "x <comp op> SOME S" имеет значение true, если значение предиката "x <comp op> s" равно true хотя бы для одного s, входящего в S. В остальных случаях значение предиката "x <comp op> SOME S" равно unknown.

Предикат exists

Предикат exists имеет следующий синтаксис:

<exists predicate> ::=
  EXISTS <subquery>

Значением этого предиката всегда является true или false, и это значение равно true тогда и только тогда, когда результат вычисления подзапроса не пуст.

15.2.3. Раздел GROUP BY

Если в табличном выражении присутствует раздел GROUP BY, то следующим выполняется он. Синтаксис раздела GROUP BY следующий:

<group by clause> ::=
  GROUP BY <column specification>
  [{,<column specification>}...]

Если обозначить через R таблицу, являющуюся результатом предыдущего раздела (FROM или WHERE), то результатом раздела GROUP BY является разбиение R на множество групп строк, состоящего из минимального числа групп таких, что для каждого столбца из списка столбцов раздела GROUP BY во всех строках каждой группы, включающей более одной строки, значения этого столбца равны. Для обозначения результата раздела GROUP BY в стандарте используется термин "сгруппированная таблица".

Раздел HAVING

Наконец, последним при вычислении табличного выражения используется раздел HAVING (если он присутствует). Синтаксис этого раздела следующий:

<having clause> ::=
  HAVING <search condition>

Раздел HAVING может осмысленно появиться в табличном выражении только в том случае, когда в нем присутствует раздел GROUP BY. Условие поиска этого раздела задает условие на группу строк сгруппированной таблицы. Формально раздел HAVING может присутствовать и в

табличном выражении, не содержащем GROUP BY. В этом случае полагается, что результат вычисления предыдущих разделов представляет собой сгруппированную таблицу, состоящую из одной группы без выделенных столбцов группирования.

Условие поиска раздела HAVING строится по тем же синтаксическим правилам, что и условие поиска раздела WHERE, и может включать те же самые предикаты. Однако имеются специальные синтаксические ограничения по части использования в условии поиска спецификаций столбцов таблиц из раздела FROM данного табличного выражения. Эти ограничения следуют из того, что условие поиска раздела HAVING задает условие на целую группу, а не на индивидуальные строки.

Поэтому в арифметических выражениях предикатов, входящих в условие выборки раздела HAVING, прямо можно использовать только спецификации столбцов, указанных в качестве столбцов группирования в разделе GROUP BY. Остальные столбцы можно специфицировать только внутри спецификаций агрегатных функций COUNT, SUM, AVG, MIN и MAX, вычисляющих в данном случае некоторое агрегатное значение для всей группы строк. Аналогично обстоит дело с подзапросами, входящими в предикаты условия выборки раздела HAVING: если в подзапросе используется характеристика текущей группы, то она может задаваться только путем ссылки на столбцы группирования.

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.

*****Агрегатные функции и результаты запросов

Агрегатные функции (в стандарте SQL/89 они называются функциями над множествами) определяются в SQL/89 следующими синтаксическими правилами:

<set function specification> ::=
  COUNT(*) | <distinct set function>
        | <all set function>
<distinct set function> ::=
  { AVG | MAX | MIN | SUM | COUNT }
  (DISTNICT <column specification>)
<all set function> ::=
  { AVG | MAX | MIN | SUM } ([ALL] <value expression>)

Как видно из этих правил, в стандарте SQL/89 определены пять стандартных агрегатных функций:

COUNT - число строк или значений,
MAX - максимальное значение,
MIN - минимальное значение,
SUM - суммарное значение
AVG - среднее значение.

Семантика агрегатных функций

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством строк может быть группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. Для всех агрегатных функций, кроме COUNT(*), фактический (т.е. требуемый семантикой) порядок вычислений следующий: на основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции COUNT для него есть 0, а значение всех остальных функций - null.

Пусть T обозначает тип значений из этого списка. Тогда результат вычисления функции COUNT - точное число с масштабом и точностью, определяемыми в реализации. Тип результата значений функций MAX и MIN совпадает с T. При вычислении функций SUM и AVG тип T не должен быть типом символьных строк, а тип результата функции - это тип

точных чисел с определяемыми в реализации масштабом и точностью, если T - тип точных чисел, и тип приблизительных чисел с определяемой в реализации точностью, если T - тип приблизительных чисел.

Вычисление функции COUNT(*) производится путем подсчета числа строк в заданном множестве. Все строки считаются различными, даже если они состоят из одного столбца со значением null во всех строках.

Если агрегатная функция специфицирована с ключевым словом DISTINCT, то список значений строится из значений указанного столбца. (Подчеркнем, что в этом случае не допускается вычисление арифметических выражений!) Далее из этого списка удаляются неопределенные значения, и в нем устраняются значения-дубликаты. Затем вычисляется указанная функция.

Если агрегатная функция специфицирована без ключевого слова DISTINCT (или с ключевым словом ALL), то список значений формируется из значений арифметического выражения, вычисляемого для каждой строки заданного множества. Далее из списка удаляются неопределенные значения, и производится вычисление агрегатной функции. Обратите внимание, что в этом случае не допускается применение функции COUNT!

Замечание: оба ограничения, указанные в двух предыдущих абзацах, являются более техническими, чем принципиальными, и могут отсутствовать в конкретных реализациях. Тем не менее, это ограничения стандарта SQL/89, и их нужно придерживаться при мобильном программировании.

Агрегатные функции можно разумно использовать в спецификации курсора, операторе выборки и подзапросе после ключевого слова SELECT (будем называть в этом подразделе все такие конструкции списком выборки, не забывая о том, что в случае подзапроса этот список состоит только из одного элемента), и в условии выборки раздела HAVING. Стандарт допускает более экзотические использования агрегатных функций в подзапросах (агрегатная функция на группе кортежей внешнего запроса), но на практике они встречаются очень редко.

База данных

Создание базы данных

В различных СУБД процедура создания *баз данных* обычно закрепляется только за администратором *баз данных*. В однопользовательских системах принимаемая по умолчанию *база данных* может быть сформирована непосредственно в процессе установки и настройки самой СУБД. Стандарт SQL не определяет, как должны создаваться *базы данных*, поэтому в каждом из диалектов языка SQL обычно используется свой подход. В соответствии со стандартом SQL, *таблицы* и другие объекты *базы данных* существуют в некоторой среде. Помимо всего прочего, каждая среда состоит из одного или более *каталогов*, а каждый *каталог* – из набора *схем*. *Схема* представляет собой поименованную коллекцию объектов *базы данных*, некоторым образом связанных друг с другом (все объекты в *базе данных* должны быть описаны в той или иной *схеме*). Объектами *схемы* могут быть *таблицы*, представления, домены, утверждения, сопоставления, толкования и наборы символов. Все они имеют одного и того же владельца и множество общих значений, принимаемых по умолчанию.
Стандарт SQL оставляет за разработчиками СУБД право выбора конкретного механизма создания и уничтожения *каталогов*, однако механизм создания и удаления *схем* регламентируется посредством операторов CREATE SCHEMA и DROP SCHEMA. В стандарте также указано, что в рамках оператора создания *схемы* должна существовать возможность определения диапазона привилегий, доступных пользователям создаваемой *схемы*. Однако конкретные способы определения подобных привилегий в разных СУБД различаются.
В настоящее время операторы CREATE SCHEMA и DROP SCHEMA реализованы в очень немногих СУБД. В других реализациях, например, в СУБД MS SQL Server, используется оператор CREATE DATABASE.

Создание базы данных в среде MS SQL Server

Процесс создания *базы данных* в системе SQL-сервера состоит из двух этапов: сначала организуется сама *база данных*, а затем принадлежащий ей *журнал транзакций*. Информация размещается в соответствующих файлах, имеющих расширения *.mdf (для *базы данных*) и *.ldf. (для *журнала транзакций*). В файле *базы данных* записываются сведения об основных объектах (*таблицах*, *индексах*, просмотрах и т.д.), а в файле *журнала транзакций* – о процессе работы с транзакциями (контроль целостности данных, состояния *базы данных* до и после выполнения транзакций).
Создание *базы данных* в системе SQL-сервер осуществляется командой CREATE DATABASE. Следует отметить, что процедура создания *базы данных* в SQL-сервере требует наличия прав администратора сервера.

```
<определение_базы_данных> ::=
   CREATE DATABASE имя_базы_данных
   [ON [PRIMARY]
   [ <определение_файла>  [,...n] ]
   [,<определение_группы> [,...n] ] ]
   [ LOG ON {<определение_файла>[,...n] } ]
   [ FOR LOAD | FOR ATTACH ]
```

Рассмотрим основные параметры представленного оператора.
При выборе имени *базы данных* следует руководствоваться общими правилами именования объектов. Если имя *базы данных* содержит пробелы или любые другие недопустимые символы, оно заключается в ограничители (двойные кавычки или квадратные скобки). Имя *базы данных* должно быть уникальным в пределах сервера и не может превышать 128 символов.
При создании и изменении *базы данных* можно указать имя файла, который будет для нее создан, изменить имя, путь и исходный размер этого файла. Если в процессе использования *базы данных* планируется ее размещение на нескольких дисках, то можно создать так

называемые *вторичные файлы базы данных* с расширением *.ndf. В этом случае основная информация о *базе данных* располагается в *первичном* (PRIMARY) файле, а при нехватке для него свободного места добавляемая информация будет размещаться во *вторичном файле*. Подход, используемый в SQL-сервере, позволяет распределять содержимое *базы данных* по нескольким дисковым томам.

Параметр ON определяет список файлов на диске для размещения информации, хранящейся в *базе данных*.

Параметр PRIMARY определяет *первичный файл*. Если он опущен, то *первичным* является первый файл в списке.

Параметр LOG ON определяет список файлов на диске для размещения *журнала транзакций*. Имя файла для *журнала транзакций* генерируется на основе имени *базы данных*, и в конце к нему добавляются символы _log.

При создании *базы данных* можно определить набор файлов, из которых она будет состоять. Файл определяется с помощью следующей конструкции:

```
<определение_файла>::=
  ([ NAME=логическое_имя_файла,]
  FILENAME='физическое_имя_файла'
  [,SIZE=размер_файла ]
  [,MAXSIZE={max_размер_файла |UNLIMITED } ]
  [, FILEGROWTH=величина_прироста ] )[,...n]
```

Здесь ***логическое имя файла*** – это имя файла, под которым он будет опознаваться при выполнении различных SQL-команд.

Физическое имя файла предназначено для указания полного пути и названия соответствующего физического файла, который будет создан на жестком диске. Это имя останется за файлом на уровне операционной системы.

Параметр SIZE определяет первоначальный размер файла; минимальный размер параметра – 512 Кб, если он не указан, по умолчанию принимается 1 Мб.

Параметр MAXSIZE определяет максимальный размер файла *базы данных*. При значении параметра UNLIMITED максимальный размер *базы данных* ограничивается свободным местом на диске.

При создании *базы данных* можно разрешить или запретить автоматический рост ее размера (это определяется параметром FILEGROWTH) и указать приращение с помощью абсолютной величины в Мб или процентным соотношением. Значение может быть указано в килобайтах, мегабайтах, гигабайтах, терабайтах или процентах (%). Если указано число без суффикса МБ, КБ или %, то по умолчанию используется значение МБ. Если размер шага роста указан в процентах (%), размер увеличивается на заданную часть в процентах от размера файла. Указанный размер округляется до ближайших 64 КБ.

Дополнительные файлы могут быть включены в группу:

```
<определение_группы>::=FILEGROUP имя_группы_файлов
   <определение_файла>[,...n]
```

Пример 3.1. Создать *базу данных*, причем для данных определить три файла на диске С, для *журнала транзакций* – два файла на диске С.

```
CREATE DATABASE Archive
ON PRIMARY ( NAME=Arch1,
  FILENAME='c:\user\data\archdat1.mdf',
SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch2,
  FILENAME='c:\user\data\archdat2.mdf',
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch3,
  FILENAME='c:\user\data\archdat3.mdf',
```

```
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
LOG ON
(NAME=Archlog1,
  FILENAME='c:\user\data\archlog1.ldf',
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Archlog2,
  FILENAME='c:\user\data\archlog2.ldf',
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
```

Пример 3.1. Создание базы данных. (html, txt)

Изменение базы данных

Большинство действий по изменению конфигурации *базы данных* выполняется с помощью следующей конструкции:

```
<изменение_базы_данных> ::=
  ALTER DATABASE имя_базы_данных
  { ADD FILE <определение_файла>[,...n]
    [TO FILEGROUP имя_группы_файлов ]
  | ADD LOG FILE <определение_файла>[,...n]
  | REMOVE FILE логическое_имя_файла
  | ADD FILEGROUP имя_группы_файлов
  | REMOVE FILEGROUP имя_группы_файлов
  | MODIFY FILE <определение_файла>
  | MODIFY FILEGROUP имя_группы_файлов
  <свойства_группы_файлов>}
```

Как видно из синтаксиса, за один вызов команды может быть изменено не более одного параметра конфигурации *базы данных*. Если необходимо выполнить несколько изменений, придется разбить процесс на ряд отдельных шагов.

В *базу данных* можно добавить (ADD) новые файлы данных (в указанную группу файлов или в группу, принятую по умолчанию) или файлы *журнала транзакций*.

Параметры файлов и групп файлов можно изменять (MODIFY).

Для удаления из *базы данных* файлов или групп файлов используется параметр REMOVE. Однако удаление файла возможно лишь при условии его освобождения от данных. В противном случае сервер не разрешит удаление.

В качестве свойств группы файлов используются следующие:

READONLY – группа файлов используется только для чтения; READWRITE – в группе файлов разрешаются изменения; DEFAULT – указанная группа файлов принимается по умолчанию.

Удаление базы данных

Удаление *базы данных* осуществляется командой:

```
  DROP DATABASE имя_базы_данных [,...n]
```

Удаляются все содержащиеся в *базе данных* объекты, а также файлы, в которых она размещается. Для исполнения операции удаления *базы данных* пользователь должен обладать соответствующими правами.

Таблица

Создание таблицы

После создания общей структуры *базы данных* можно приступить к *созданию таблиц*, которые представляют собой отношения, входящие в состав проекта *базы данных*.

Таблица – основной объект для хранения информации в реляционной *базе данных*. Она состоит из содержащих данные *строк* и *столбцов*, занимает в *базе данных* физическое пространство и может быть постоянной или временной.

Поле, также называемое в реляционной *базе данных столбцом*, является частью *таблицы*, за которой закреплен определенный тип данных. Каждая *таблица базы данных* должна содержать хотя бы один *столбец*. *Строка* данных – это запись в *таблице базы данных*, она включает поля, содержащие данные из одной записи *таблицы*.

Приступая к *созданию таблицы*, необходимо иметь ответы на ряд вопросов:

- Как будет называться *таблица*?
- Как будут называться *столбцы* (поля) *таблицы*?
- Какие типы данных будут закреплены за каждым *столбцом*?
- Какой размер памяти должен быть выделен для хранения каждого *столбца*?
- Какие *столбцы таблицы* требуют обязательного ввода?
- Из каких *столбцов* будет состоять первичный ключ?

Базовый синтаксис оператора *создания таблицы* имеет следующий вид:

```
<определение_таблицы> ::=
  CREATE TABLE имя_таблицы
  (имя_столбца тип_данных
    [NULL | NOT NULL ] [,...n])
```

Приведенный стандарт совпадает с реализацией оператора *создания таблицы* в среде MS SQL Server.

Главное в команде *создания таблицы* – определение *имени таблицы* и описание набора имен полей, которые указываются в соответствующем порядке. Кроме того, этой командой оговариваются типы данных и размеры полей *таблицы*.

Ключевое слово NULL используется для указания того, что в данном *столбце* могут содержаться значения NULL. Значение NULL отличается от пробела или нуля – к нему прибегают, когда необходимо указать, что данные недоступны, опущены или недопустимы. Если указано ключевое слово NOT NULL, то будут отклонены любые попытки поместить значение NULL в данный *столбец*. Если указан параметр NULL, помещение значений NULL в *столбец* разрешено. По умолчанию стандарт SQL предполагает наличие ключевого слова NULL.

Мы использовали упрощенную версию оператора CREATE TABLE стандарта SQL. Его полная версия приводится при обсуждении вопросов обеспечения целостности данных.

Пример 3.2. Создать *таблицу* для хранения данных о товарах, поступающих в продажу в некоторой торговой фирме. Необходимо учесть такие сведения, как название и тип товара, его цена, сорт и город, где товар производится.

```
CREATE TABLE Товар
(Название      VARCHAR(50) NOT NULL,
   Цена        MONEY NOT     NULL,
   Тип         VARCHAR(50) NOT NULL,
   Сорт        VARCHAR(50),
   ГородТовара VARCHAR(50))
```

Пример 3.2. Создание таблицы для хранения данных о товарах, поступающих в продажу в некоторой торговой фирме. (html, txt)

Пример 3.3.Создать *таблицу* для сохранения сведений о постоянных клиентах с указанием названий города и фирмы, фамилии, имени и отчества клиента, номера его телефона.

```
CREATE TABLE  Клиент
   (Фирма       VARCHAR(50) NOT NULL,
   Фамилия      VARCHAR(50) NOT NULL,
   Имя          VARCHAR(50) NOT NULL,
   Отчество     VARCHAR(50),
   ГородКлиента VARCHAR(50),
   Телефон      CHAR(10) NOT NULL)
```

Пример 3.3. Создание таблицы для сохранения сведений о постоянных клиентах. (html, txt)

Изменение таблицы

Структура существующей *таблицы* может быть модифицирована с помощью команды ALTER TABLE, упрощенный синтаксис которой представлен ниже:

```
ALTER TABLE имя_таблицы
  {[ADD [COLUMN] имя_столбца тип_данных [
   NULL | NOT NULL ]]
  | [DROP [COLUMN] имя_столбца]}
```

В среде MS SQL Server упрощенный синтаксис команды модификации *таблицы* имеет вид:

```
ALTER TABLE имя_таблицы
  {[ALTER COLUMN имя_столбца
  {новый_тип_данных [(точность[,масштаб])]
   [ NULL | NOT NULL ]}]
  | ADD { [имя_столбца тип_данных]
  | имя_столбца AS выражение } [,...n]
  | DROP {COLUMN имя_столбца}[,...n]
  }
```

Команда позволяет добавлять и удалять *столбцы*, изменять их определения.

Одно из основных правил при добавлении *столбцов* в существующую *таблицу* гласит: когда в *таблице* уже содержатся данные, добавляемый *столбец* не может быть определен с атрибутом NOT NULL. Этот атрибут означает, что для каждой *строки* данных соответствующий *столбец* должен содержать некоторое значение, поэтому добавление *столбца* с атрибутом NOT NULL приводит к появлению противоречия – уже существующие *строки* данных *таблицы* не будут иметь в новом *столбце* ненулевых значений.

Тем не менее существует способ добавления обязательных полей в существующую *таблицу*. Для этого необходимо:

- добавить в *таблицу* новый *столбец*, определив его с атрибутом NULL (т.е. *столбец* не обязан содержать каких-либо значений);
- ввести в новый *столбец* какие-либо значения для каждой *строки* данных *таблицы*;
- убедившись, что новый *столбец* содержит ненулевые значения для каждой *строки данных*, изменить структуру *таблицы*, заменив атрибут этого *столбца* на NOT NULL.

При *изменении* определений *столбцов* следует принимать во внимание некоторые общепринятые правила:

- размер *столбца* может быть увеличен до максимального значения, допускаемого соответствующим типом данных;
- размер *столбца* может быть уменьшен только в том случае, если содержащееся в нем наибольшее значение не будет превосходить его нового размера;

- количество разрядов числового типа данных всегда может быть увеличено;
- количество разрядов числового типа данных может быть уменьшено только в том случае, если количество разрядов наибольшего значения в соответствующем *столбце* не будет превосходить нового числа разрядов, определенного для этого *столбца*;
- количество десятичных знаков числового типа данных может быть уменьшено или увеличено;
- *тип данных столбца*, как правило, может быть изменен.

Некоторые реализации фактически могут ограничить разработчика в использовании некоторых опций команды ALTER TABLE. Например, может оказаться недопустимым удаление *столбцов* из существующей *таблицы*. Чтобы добиться этого, сначала потребуется удалить саму *таблицу* и только потом заново ее построить с нужными *столбцами*. Причем уже внесенные в *таблицу* данные будут потеряны.
Возможны трудности, связанные с удалением из *таблицы столбца*, который зависит от некоторого *столбца* другой *таблицы*. В таком случае сначала придется удалить ограничение *столбца*, а затем сам *столбец*.
Пример 3.4. Добавить в *таблицу* Клиент поле для номера расчетного счета.

```
ALTER TABLE Клиент ADD Рас_счет CHAR(20)
```

Пример 3.4. Добавление в таблицу Клиент поля для номера расчетного счета. (html, txt)

Удаление таблицы

С течением времени структура *базы данных* меняется: создаются новые *таблицы*, а прежние становятся ненужными и удаляются из *базы данных* с помощью оператора:

```
DROP TABLE имя_таблицы [RESTRICT | CASCADE]
```

Следует отметить, что эта команда удалит не только указанную *таблицу*, но и все входящие в нее *строки* данных. Если требуется удалить из *таблицы* лишь данные, сохранив структуру *таблицы*, следует воспользоваться командой DELETE.
Оператор DROP TABLE дополнительно позволяет указывать, следует ли операцию *удаления* выполнять каскадно. Если в операторе указано ключевое слово RESTRICT, то при наличии в *базе данных* хотя бы одного объекта, существование которого зависит от удаляемой *таблицы*, выполнение оператора DROP TABLE будет отменено. Если указано ключевое слово CASCADE, автоматически удаляются и все прочие объекты *базы данных*, чье существование зависит от удаляемой *таблицы*, а также другие объекты, зависящие от удаляемых объектов. Общий эффект от выполнения оператора DROP TABLE с ключевым словом CASCADE может оказаться весьма ощутимым, поэтому подобные операторы следует использовать с максимальной осторожностью.
Чаще всего оператор DROP TABLE используется для исправления ошибок, допущенных при *создании таблицы*. Если *таблица* была создана с некорректной структурой, можно воспользоваться оператором DROP TABLE для ее *удаления*, после чего создать *таблицу* заново.

Индексы

Индексы в стандарте языка

Индексы представляют собой структуру, позволяющую выполнять ускоренный доступ к *строкам таблицы* на основе значений одного или более ее *столбцов*. Наличие *индекса* может существенно повысить скорость выполнения некоторых запросов и сократить время поиска необходимых данных за счет физического или логического их упорядочивания. ***Индекс*** – это набор ссылок, упорядоченных по определенному *столбцу таблицы*, который в данном случае будет называться индексированным *столбцом*. Хотя *индекс* и связан с конкретным *столбцом* (или *столбцами*) *таблицы*, все же он является самостоятельным объектом *базы данных*.

Физически *индекс* – всего лишь упорядоченный набор значений из индексированного *столбца* с указателями на места физического размещения исходных *строк* в структуре *базы данных*. Когда пользователь выполняет обращающийся к индексированному *столбцу* запрос, СУБД автоматически анализирует *индекс* для поиска требуемых значений.
Однако, поскольку *индексы* должны обновляться системой при каждом внесении *изменений* в их базовую *таблицу*, они создают дополнительную нагрузку на систему.
Индексы обычно создаются с целью удовлетворения определенных критериев поиска после того, как *таблица* уже находилась некоторое время в работе и увеличилась в размерах.
Создание индексов не предусмотрено стандартом SQL, однако большинство диалектов поддерживают как минимум следующий оператор:

```
CREATE [ UNIQUE ] INDEX имя_индекса
  ON имя_таблицы(имя_столбца[ASC|DESC][,...n])
```

Указанные в операторе *столбцы* составляют *ключ индекса*. *Индексы* могут создаваться только для базовых *таблиц*, но не для представлений. Если в операторе указано ключевое слово UNIQUE, уникальность значений *ключа индекса* будет автоматически поддерживаться системой. Требование уникальности значений обязательно для первичных ключей, а также возможно и для других *столбцов таблицы* (например, для альтернативных ключей). Хотя *создание индекса* допускается в любой момент, при его построении для уже заполненной данными *таблицы* могут возникнуть проблемы, связанные с дублированием данных в различных *строках*. Следовательно, *уникальные индексы* (по крайней мере, для первичного ключа) имеет смысл создавать непосредственно при формировании *таблицы*. В результате система сразу возьмет на себя контроль за уникальностью значений данных в соответствующих *столбцах*.
Если созданный *индекс* впоследствии окажется ненужным, его можно удалить с помощью оператора

```
DROP INDEX имя_индекса
```

Индексы в среде MS SQL Server

Индекс представляет собой средство, помогающее ускорить поиск необходимых данных за счет физического или логического их упорядочивания. Индекс представляет собой набор ссылок, упорядоченных по определенному столбцу таблицы, который в данном случае будет называться индексированным столбцом. Индексы - это наборы уникальных значений для некоторой *таблицы* с соответствующими ссылками на данные. Они расположены в самой *таблице* и являются удобным внутренним механизмом системы SQL-сервера, с помощью которого осуществляется доступ к данным наиболее оптимальным способом. В среде SQL Server реализованы эффективные алгоритмы поиска нужного значения в строго определенной последовательности данных. Ускорение поиска достигается именно за счет того, что данные представляются упорядоченными (хотя физически, в зависимости от типа *индекса*, они могут храниться в соответствии с очередностью их добавления в *таблицу*). К настоящему времени разработаны эффективные математические алгоритмы поиска данных в упорядоченной последовательности. Наиболее эффективной структурой для поиска данных в машинном представлении являются B-деревья – многоуровневая иерархическая структура с переменным количеством элементов в каждом узле.

Создание индекса

Если выборка данных из *таблицы* требует значительного времени, это означает, что для нее необходимо создать *индекс*. *Индексы* могут существенно повысить производительность выполнения операций поиска и выборки данных. При выборе *столбца* для *индекса* следует проанализировать, какие типы запросов чаще всего выполняются пользователями и какие *столбцы* являются ключевыми, т.е. задающими критерии выборки данных, например, порядок сортировки.

В среде SQL Server реализовано несколько типов *индексов*:

- *кластерные индексы*;
- *некластерные индексы*;
- *уникальные индексы*.

Некластерный индекс

Некластерные индексы – наиболее типичные представители семейства *индексов*. В отличие от *кластерных*, они не перестраивают физическую структуру *таблицы*, а лишь организуют ссылки на соответствующие *строки*.

Для идентификации нужной *строки* в *таблице некластерный индекс* организует специальные указатели, включающие в себя:

- информацию об идентификационном номере файла, в котором хранится *строка*;
- идентификационный номер страницы соответствующих данных;
- номер искомой *строки* на соответствующей странице;
- содержимое *столбца*.

В большинстве случаев следует ограничиваться 4-5 *индексами*.

Кластерный индекс

Принципиальным отличием ***кластерного индекса*** от *индексов* других типов является то, что при его определении в *таблице* физическое расположение данных перестраивается в соответствии со структурой *индекса*. Логическая структура *таблицы* в этом случае представляет собой скорее словарь, чем *индекс*. Данные в словаре физически упорядочены, например по алфавиту.

Кластерные индексы могут дать существенное увеличение производительности поиска данных даже по сравнению с обычными *индексами*. Увеличение производительности особенно заметно при работе с последовательными данными. Если в *таблице* определен *некластерный индекс*, то сервер должен сначала обратиться к *индексу*, а затем найти нужную *строку* в *таблице*. При использовании *кластерных индексов* следующая порция данных располагается сразу после найденных ранее данных. Благодаря этому отпадают лишние операции, связанные с обращением к *индексу* и новым поиском нужной *строки* в *таблице*.

Естественно, в *таблице* может быть определен только один *кластерный индекс*. В качестве такового следует выбирать наиболее часто используемые *столбцы*. При этом стоит следовать общим рекомендациям *создания индексов* и не индексировать слишком длинные *столбцы*. *Кластерный индекс* может включать несколько *столбцов*. Однако количество таких *столбцов* рекомендуется по возможности свести к минимуму.

Необходимо избегать *создания кластерного индекса* для часто изменяемых *столбцов*, поскольку сервер должен будет выполнять физическое перемещение всех данных в *таблице*, чтобы они находились в упорядоченном состоянии, как того требует *кластерный индекс*. Для интенсивно изменяемых *столбцов* лучше подходит *некластерный индекс*.

При создании в *таблице* первичного ключа (PRIMARY KEY) сервер автоматически создает для него *кластерный индекс*, если его не существовало ранее или если при определении ключа не был явно указан другой тип *индекса*.

Когда же в *таблице* определен еще и *некластерный индекс*, то его указатель ссылается не на физическое положение *строки* в *базе данных*, а на соответствующий элемент *кластерного индекса*, описывающего эту *строку*, что позволяет не перестраивать структуру *некластерных индексов* всякий раз, когда *кластерный индекс* меняет физический порядок *строк* в *таблице*.

Уникальный индекс

Уникальность значений в индексируемом *столбце* гарантируют *уникальные индексы*. При их наличии сервер не разрешит вставить новое или изменить существующее значение таким образом, чтобы в результате этой операции в *столбце* появились два одинаковых значения. *Уникальный индекс* является своеобразной надстройкой и может быть реализован как для *кластерного*, так и для *некластерного индекса*. В одной *таблице* может существовать один уникальный *кластерный* и множество уникальных *некластерных индексов*.

Уникальные индексы следует определять только тогда, когда это действительно необходимо. Для обеспечения целостности данных в *столбце* можно определить ограничение целостности UNIQUE или PRIMARY KEY, а не прибегать к *уникальным индексам*. Их использование только для обеспечения целостности данных является неоправданной тратой пространства в *базе данных*. Кроме того, на их поддержание тратится и процессорное время.

Средства языка SQL предлагают несколько способов определения *индекса*:

- автоматическое *создание индекса* при создании первичного ключа;
- автоматическое *создание индекса* при определении ограничения целостности UNIQUE;
- *создание индекса* с помощью команды CREATE INDEX.

Последняя команда имеет следующий формат:

```
<создание_индекса>::=
  CREATE [ UNIQUE ]
    [ CLUSTERED | NONCLUSTERED ]
  INDEX имя_индекса ON имя_таблицы(имя_столбца
    [ASC|DESC][,...n])
  [WITH [PAD_INDEX]
    [[,] FILLFACTOR=фактор_заполнения]
    [[,] IGNORE_DUP_KEY]
    [[,] DROP_EXISTING]
    [[,] STATISTICS_NORECOMPUTE] ]
    [ON имя_группы_файлов ]
```

Рассмотрим некоторые параметры приведенной команды.

Имя *индекса* должно быть уникальным в пределах *таблицы*, а сам *индекс* создается исключительно для *таблицы* текущей *базы данных*.

Параметр UNIQUE используется при необходимости ввода в определенное поле только уникальных значений. При указании этого ключевого слова будет создан *уникальный индекс*. В индексируемом *столбце* желательно запретить хранение значений NULL, чтобы избежать проблем, связанных с уникальностью значений. После того как для *столбца* появится *уникальный индекс*, сервер не разрешит выполнение команд INSERT и UPDATE, которые приведут к появлению дублирующих значений.

Параметр CLUSTERED использует возможность физического индексирования данных и позволяет произвести так называемое кластерное индексирование, в результате чего будут отсортированы данные в самой *таблице* согласно порядку этого *индекса*, а вся добавляемая информация станет приводить к изменению физического порядка данных. *Кластерным* может быть только один *индекс* в *таблице*.

Параметр NONCLUSTERED позволяет создавать *некластерные индексы*.

Параметр FILLFACTOR осуществляет настройку разбиения *индекса* на страницы и заметно оптимизирует работу SQL-сервера. Коэффициент FILLFACTOR определяет в процентном соотношении размер создаваемых индексных страниц. При этом имеется обратно пропорциональная зависимость частоты работы с *таблицей* и коэффициента FILLFACTOR.

Параметр PAD_INDEX определяет заполнение внутреннего пространства *индекса* и применяется совместно с FILLFACTOR.

Параметр DROP_EXISTING при использовании *кластерного индекса* определяет его повторное создание, что позволяет предотвратить нежелательное обновление *кластерных индексов*.
Параметр STATISTICS_NORECOMPUTE определяет функции автоматического обновления статистики для *таблицы*.
Параметр имя_группы_файлов позволяет осуществить выбор файловой группы, в которой будет находиться создаваемый *индекс*. Использование *индекса* из другой файловой группы повышает производительность *некластерных индексов* в связи с параллельностью выполнения процессов ввода/вывода и работы с самим *индексом*.

Удаление индекса

Удаление индекса выполняется командой

```
DROP INDEX 'имя_индекса'[,...n]
```

Пример 3.5. Создать уникальный *кластерный индекс* для *таблицы* Клиент по *столбцу* Фамилия в первичной группе файлов.

```
CREATE UNIQUE CLUSTERED INDEX index_klient1
  ON Клиент (Фамилия)
  WITH DROP_EXISTING
  ON PRIMARY
```

Пример 3.5. Создание уникального кластерного индекса. (html, txt)

Пример 3.6. Создать уникальный *некластерный индекс* для *таблицы* Клиент по *столбцам* Фамилия и Имя в первичной группе файлов. Кроме того, элементы *индекса* будут упорядочены по убыванию. Также запретим автоматическое обновление статистики при изменении данных в *таблице* и установим фактор заполнения индексных страниц на уровне 30%.

```
CREATE UNIQUE NONCLUSTERED INDEX index_klient2
  ON Клиент (Фамилия DESC,Имя DESC)
  WITH FILLFACTOR=30,
  STATISTICS_NORECOMPUTE
  ON PRIMARY
```

Пример 3.6. Создание уникального некластерного индекса. (html, txt)

Лекция: Функции пользователя

1 Понятие функции пользователя

При реализации на языке SQL сложных алгоритмов, которые могут потребоваться более одного раза, сразу встает вопрос о сохранении разработанного кода для дальнейшего применения. Эту задачу можно было бы реализовать с помощью *хранимых процедур*, однако их архитектура не позволяет использовать процедуры непосредственно в выражениях, т.к. они требуют промежуточного присвоения возвращенного значения переменной, которая затем и указывается в выражении. Естественно, подобный метод применения программного кода не слишком удобен. Многие разработчики уже давно хотели иметь возможность вызова разработанных алгоритмов непосредственно в выражениях.
Возможность создания *пользовательских функций* была предоставлена в среде MS SQL Server 2000. В других реализациях SQL в распоряжении пользователя имеются только *встроенные*

функции, которые обеспечивают выполнение наиболее распространенных алгоритмов: поиск максимального или минимального значения и др.
Функции пользователя представляют собой самостоятельные объекты базы данных, такие, например, как *хранимые процедуры* или триггеры. *Функция пользователя* располагается в определенной базе данных и доступна только в ее контексте.
В SQL Server имеются следующие классы *функций пользователя*:

- ***Scalar*** – *функции* возвращают обычное скалярное значение, каждая может включать множество команд, объединяемых в один блок с помощью конструкции BEGIN...END;
- **Inline** – *функции* содержат всего одну команду SELECT и возвращают пользователю набор данных в виде значения *типа данных TABLE*;
- ***Multi-statement*** – *функции* также возвращают пользователю значение *типа данных TABLE*, содержащее набор данных, однако в теле *функции* находится множество команд SQL (INSERT, UPDATE и т.д.). Именно с их помощью и формируется набор данных, который должен быть возвращен после выполнения *функции*.

Пользовательские функции сходны с *хранимыми процедурами*, но, в отличие от них, могут применяться в запросах так же, как и системные *встроенные функции*. *Пользовательские функции*, возвращающие таблицы, могут стать альтернативой просмотрам. Просмотры ограничены одним выражением SELECT, а *пользовательские функции* способны включать дополнительные выражения, что позволяет создавать более сложные и мощные конструкции.

2 Функции Scalar

Создание и изменение *функции* данного типа выполняется с помощью команды:

```
<определение_скаляр_функции>::=
{CREATE | ALTER } FUNCTION [владелец.]
   имя_функции
( [ { @имя_параметра скаляр_тип_данных
   [=default]}[,...n]])
RETURNS скаляр_тип_данных
[WITH {ENCRYPTION | SCHEMABINDING}
   [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN скаляр_выражение
END
```

Рассмотрим назначение параметров команды.
Функция может содержать один или несколько *входных параметров* либо не содержать ни одного. Каждый параметр должен иметь уникальное в пределах создаваемой *функции* имя и начинаться с символа "@". После имени указывается тип данных параметра. Дополнительно можно указать значение, которое будет автоматически присваиваться параметру (DEFAULT), если пользователь явно не указал значение соответствующего параметра при вызове *функции*.
С помощью конструкции RETURNS скаляр_тип_данных указывается, какой тип данных будет иметь возвращаемое *функцией* значение.
Дополнительные параметры, с которыми должна быть создана *функция*, могут быть указаны посредством ключевого слова WITH. Благодаря ключевому слову ENCRYPTION код команды, используемый для создания *функции*, будет зашифрован, и никто не сможет просмотреть его. Эта возможность позволяет скрыть логику работы *функции*. Кроме того, в теле *функции* может выполняться обращение к различным объектам базы данных, а потому изменение или удаление

соответствующих объектов может привести к нарушению работы *функции*. Чтобы избежать этого, требуется запретить внесение изменений, указав при создании этой *функции* ключевое слово SCHEMABINDING.
Между ключевыми словами BEGIN...END указывается набор команд, они и будут являться телом *функции*.
Когда в ходе выполнения кода *функции* встречается ключевое слово RETURN, выполнение *функции* завершается и как результат ее вычисления возвращается значение, указанное непосредственно после слова RETURN. Отметим, что в теле *функции* разрешается использование множества команд RETURN, которые могут возвращать различные значения. В качестве возвращаемого значения допускаются как обычные константы, так и сложные выражения. Единственное условие – тип данных возвращаемого значения должен совпадать с типом данных, указанным после ключевого слова RETURNS.
Пример 11.1. Создать и применить функцию скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату. Владелец функции – пользователь с именем user1.

```
CREATE FUNCTION
   user1.sales(@data DATETIME)
RETURNS INT
AS
BEGIN
DECLARE @c INT
SET @c=(SELECT SUM(количество)
      FROM Сделка
      WHERE дата=@data)
RETURN (@c)
END
```

Пример 11.1. Создание функции скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату. (html, txt)

В качестве *входного параметра* используется дата. *Функция* возвращает значение целого типа, полученное из оператора SELECT путем суммирования количества товара из таблицы Сделка. Условием отбора записей для суммирования является равенство даты сделки значению *входного параметра функции*.
Проиллюстрируем обращение к *функции пользователя*: определим количество товара, поступившего за 02.11.01:

```
DECLARE @kol INT
SET @kol=user1.sales ('02.11.01')
SELECT @kol
```

3 Функции Inline

Создание и изменение *функции* этого типа выполняется с помощью команды:

```
<определение_табл_функции>::=
{CREATE | ALTER } FUNCTION [владелец.]
   имя_функции
( [ { @имя_параметра скаляр_тип_данных
   [=default]}[,...n]])
RETURNS TABLE
[ WITH {ENCRYPTION | SCHEMABINDING}
   [,...n] ]
[AS]
```

```
RETURN [(] SELECT_оператор [)]
```

Основная часть параметров, используемых при создании *табличных функций*, аналогична параметрам *скалярной функции*. Тем не менее создание *табличных функций* имеет свою специфику.

После ключевого слова RETURNS всегда должно указываться ключевое слово TABLE. Таким образом, *функция* данного типа должна строго возвращать значение *типа данных TABLE*.

Структура возвращаемого значения *типа TABLE* не указывается явно при описании собственно типа данных. Вместо этого сервер будет автоматически использовать для возвращаемого значения TABLE структуру, возвращаемую запросом SELECT, который является единственной командой *функции*.

Особенность *функции* данного типа заключается в том, что структура значения TABLE создается автоматически в ходе выполнения запроса, а не указывается явно при определении типа после ключевого слова RETURNS.

Возвращаемое *функцией* значение *типа TABLE* может быть использовано непосредственно в запросе, т.е. в разделе FROM.

Пример 11.2. Создать и применить функцию табличного типа для определения двух наименований товара с наибольшим остатком.

```
CREATE FUNCTION user1.itog()
RETURNS TABLE
AS
RETURN (SELECT TOP 2 Товар.Название
  FROM Товар INNER JOIN Склад
  ON Товар.КодТовара=Склад.КодТовара
  ORDER BY Склад.Остаток DESC)
```

Пример 11.2. Создание функции табличного типа для определения двух наименований товара с наибольшим остатком. (html, txt)

Использовать *функцию* для получения двух наименований товара с наибольшим остатком можно следующим образом:

```
SELECT Название
FROM user1.itog()
```

4 Функции Multi-statement

Создание и изменение *функций* типа *Multi-statement* выполняется с помощью следующей команды:

```
<определение_мульти_функции>::=
{CREATE | ALTER }FUNCTION [владелец.]
  имя_функции
( [ { @имя_параметра скаляр_тип_данных
  [=default]}[,...n]])
RETURNS @имя_параметра TABLE
  <определение_таблицы>
[WITH {ENCRYPTION | SCHEMABINDING}
  [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN
```

```
  END
```
Использование большей части параметров рассматривалось при описании предыдущих *функций*.
Отметим, что *функции* данного типа, как и *табличные*, возвращают значение *типа TABLE*. Однако, в отличие от *табличных функций*, при создании *функций Multi-statement* необходимо явно задать структуру возвращаемого значения. Она указывается непосредственно после ключевого слова TABLE и, таким образом, является частью определения возвращаемого типа данных. Синтаксис конструкции <определение_таблицы> полностью соответствует одноименным структурам, используемым при создании обычных таблиц с помощью команды CREATE TABLE.
Набор возвращаемых данных должен формироваться с помощью команд INSERT, выполняемых в теле *функции*. Кроме того, в теле *функции* допускается использование различных конструкций языка SQL, которые могут контролировать значения, размещаемые в выходном наборе строк. При работе с командой INSERT требуется явно указать имя того объекта, куда необходимо вставить строки. Поэтому в *функциях* типа *Multi-statement*, в отличие от *табличных*, необходимо присвоить какое-то имя объекту с *типом данных TABLE* – оно и указывается как возвращаемое значение.
Завершение работы *функции* происходит в двух случаях: если возникают ошибки выполнения и если появляется ключевое слово RETURN. В отличие от *функций скалярного типа*, при использовании команды RETURN не нужно указывать возвращаемое значение. Сервер автоматически возвратит набор данных *типа TABLE*, имя и структура которого была указана после ключевого слова RETURNS. В теле *функции* может быть указано более одной команды RETURN.
Необходимо отметить, что работа *функции* завершается только при наличии команды RETURN. Это утверждение верно и в том случае, когда речь идет о достижении конца тела *функции* – самой последней командой должна быть команда RETURN.
Пример 11.3. Создать и применить функцию (типа multi-statement), которая для некоторого сотрудника выводит список всех его подчиненных (подчиненных как непосредственно ему, так и опосредствованно через других сотрудников).
Список сотрудников с указанием каждого руководителя представлен в таблице emp_mgr со следующей структурой:
```
  CREATE TABLE emp_mgr
  (emp CHAR(2) PRIMARY KEY,-- сотрудник
  mgr CHAR(2))          -- руководитель
```
Пример данных в таблице emp_mgr показан ниже. Для упрощения иллюстрации имена сотрудников и их начальников представлены буквами латинского алфавита. У директора организации начальника нет (NULL).
```
  emp     mgr
  ---------
  a     NULL
  b     a
  c     a
  d     a
  e     f
  f     b
  g     b
  i     c
  k     d

  CREATE FUNCTION fn_findReports(@id_emp
     CHAR(2))
```

```
RETURNS @report TABLE(empid CHAR(2)
   PRIMARY KEY,
   mgrid CHAR(2))
AS
BEGIN
   DECLARE @r INT
   DECLARE @t TABLE(empid CHAR(2)
      PRIMARY KEY,
      mgrid CHAR(2),
        pr   INT DEFAULT 0)
INSERT @t SELECT emp,mgr,0
      FROM emp_mgr
      WHERE emp=@id_emp
SET @r=@@ROWCOUNT
WHILE @r>0
BEGIN
   UPDATE @t   SET pr=1   WHERE pr=0
   INSERT @t SELECT e.emp, e.mgr,0
      FROM  emp_mgr e, @t t
      WHERE e.mgr=t.empid
         AND t.pr=1
   SET @r=@@ROWCOUNT
   UPDATE @t SET pr=2 WHERE pr=1
END
INSERT @report SELECT empid, mgrid
   FROM @t
RETURN
END
```

Пример 11.3. Создание функции, которая для некоторого сотрудника выводит список всех его подчиненных. (html, txt)

Применим созданную *функцию* для определения списка подчиненных сотрудника ‘b’:

```
SELECT * FROM fn_findReports('b')
```

Оператор возвращает следующие значения:

```
emp     mgr
-----------
b     a
e     f
f     b
g     b
```

Список подчиненных сотрудника ‘a’ создается с помощью оператора

```
SELECT * FROM fn_findReports('a')

emp     mgr
---------
a     NULL
b     a
c     a
d     a
e     f
f     b
```

```
g    b
i    c
k    d
```

Другой оператор формирует список подчиненных сотрудника ‘e’:

```
SELECT * FROM fn_findReports('e')
emp    mgr
--------
e    f
```

Список подчиненных сотрудника ‘c’ создает следующий оператор:

```
SELECT * FROM fn_findReports('c')
emp    mgr
--------
c    a
i    c
```

Удаление любой *функции* осуществляется командой:

```
DROP FUNCTION {[ владелец.] имя_функции }
    [,...n]
```

5 Встроенные функции

Встроенные функции, имеющиеся в распоряжении пользователей при работе с SQL, можно условно разделить на следующие группы:

- *математические функции*;
- *строковые функции*;
- *функции для работы с датой и временем*;
- функции конфигурирования;
- функции системы безопасности;
- функции управления метаданными;
- статистические функции.

Математические функции

Краткий обзор *математических функций* представлен в таблице.

Таблица 11.1.

ABS	вычисляет абсолютное значение числа
ACOS	вычисляет арккосинус
ASIN	вычисляет арксинус
ATAN	вычисляет арктангенс
ATN2	вычисляет арктангенс с учетом квадратов
CEILING	выполняет округление вверх
COS	вычисляет косинус угла

COT	возвращает котангенс угла
DEGREES	преобразует значение угла из радиан в градусы
EXP	возвращает экспоненту
FLOOR	выполняет округление вниз
LOG	вычисляет натуральный логарифм
LOG10	вычисляет десятичный логарифм
PI	возвращает значение "пи"
POWER	возводит число в степень
RADIANS	преобразует значение угла из градуса в радианы
RAND	возвращает случайное число
ROUND	выполняет округление с заданной точностью
SIGN	определяет знак числа
SIN	вычисляет синус угла
SQUARE	выполняет возведение числа в квадрат
SQRT	извлекает квадратный корень
TAN	возвращает тангенс угла

```
SELECT Товар.Название, Сделка.Количество,
Round(Товар.Цена*Сделка.Количество
   *0.05,1)
AS Налог
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=
   Сделка.КодТовара
```

Пример 11.4. Использование функции округления до одного знака после запятой для расчета налога. (html, txt)

Строковые функции

Краткий обзор *строковых функций* представлен в таблице.

Таблица 11.2.

ASCII	возвращает код ASCII левого символа строки

CHAR	по коду ASCII возвращает символ
CHARINDEX	определяет порядковый номер символа, с которого начинается вхождение подстроки в строку
DIFFERENCE	возвращает показатель совпадения строк
LEFT	возвращает указанное число символов с начала строки
LEN	возвращает длину строки
LOWER	переводит все символы строки в нижний регистр
LTRIM	удаляет пробелы в начале строки
NCHAR	возвращает по коду символ Unicode
PATINDEX	выполняет поиск подстроки в строке по указанному шаблону
REPLACE	заменяет вхождения подстроки на указанное значение
QUOTENAME	конвертирует строку в формат Unicode
REPLICATE	выполняет тиражирование строки определенное число раз
REVERSE	возвращает строку, символы которой записаны в обратном порядке
RIGHT	возвращает указанное число символов с конца строки
RTRIM	удаляет пробелы в конце строки
SOUNDEX	возвращает код звучания строки
SPACE	возвращает указанное число пробелов
STR	выполняет конвертирование значения числового типа в символьный формат
STUFF	удаляет указанное число символов, заменяя новой подстрокой
SUBSTRING	возвращает для строки подстроку указанной длины с заданного символа
UNICODE	возвращает Unicode-код левого символа строки
UPPER	переводит все символы строки в верхний регистр

```
SELECT Фирма, [Фамилия]+""
   +Left([Имя],1)+"."
   +Left([Отчество],1)
```

```
    +"." AS ФИО
FROM Клиент
```

Пример 11.5. Использование функции LEFT для получения инициалов клиентов. (html, txt)

Функции для работы с датой и временем

Краткий обзор основных *функций для работы с датой и временем* представлен в таблице.

Таблица 11.3.

DATEADD	добавляет к дате указанное значение дней, месяцев, часов и т.д.
DATEDIFF	возвращает разницу между указанными частями двух дат
DATENAME	выделяет из даты указанную часть и возвращает ее в символьном формате
DATEPART	выделяет из даты указанную часть и возвращает ее в числовом формате
DAY	возвращает число из указанной даты
GETDATE	возвращает текущее системное время
ISDATE	проверяет правильность выражения на соответствие одному из возможных форматов ввода даты
MONTH	возвращает значение месяца из указанной даты
YEAR	возвращает значение года из указанной даты

```
SELECT Year(Дата) AS Год, Month(Дата)
  AS Месяц,
  Sum(Количество) AS Общ_Количество
FROM Сделка
GROUP BY Year(Дата), Month(Дата)
```

Пример 11.6. Использование функций YEAR и MONTH для определения общего количества товара, проданного за каждый месяц каждого года. (html, txt)

```
DECLARE @d DATETIME
DECLARE @y INT
SET @d='29.10.03'
SET @y=DATEPART(yy,@d)
SELECT @y
```

Пример 11.7. Пример выделения из даты значения года. (html, txt)

Лекция: Хранимые процедуры

1 Понятие хранимой процедуры

Хранимые процедуры представляют собой группы связанных между собой операторов SQL, применение которых делает работу программиста более легкой и гибкой, поскольку выполнить *хранимую процедуру* часто оказывается гораздо проще, чем последовательность отдельных операторов SQL. Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде. *Выполнение* в базе данных *хранимых процедур* вместо отдельных операторов SQL дает пользователю следующие преимущества:

- необходимые операторы уже содержатся в базе данных;
- все они прошли этап *синтаксического анализа* и находятся в исполняемом формате; перед *выполнением хранимой процедуры* SQL Server генерирует для нее *план исполнения*, выполняет ее оптимизацию и компиляцию;
- *хранимые процедуры* поддерживают *модульное программирование*, так как позволяют разбивать большие задачи на самостоятельные, более мелкие и удобные в управлении части;
- *хранимые процедуры* могут вызывать другие *хранимые процедуры* и функции;
- *хранимые процедуры* могут быть вызваны из прикладных программ других типов;
- как правило, *хранимые процедуры* выполняются быстрее, чем последовательность отдельных операторов;
- *хранимые процедуры* проще использовать: они могут состоять из десятков и сотен команд, но для их запуска достаточно указать всего лишь имя нужной *хранимой процедуры*. Это позволяет уменьшить размер запроса, посылаемого от клиента на сервер, а значит, и нагрузку на сеть.

Хранение процедур в том же месте, где они исполняются, обеспечивает уменьшение объема передаваемых по сети данных и повышает общую производительность системы. Применение *хранимых процедур* упрощает сопровождение программных комплексов и внесение изменений в них. Обычно все ограничения целостности в виде правил и алгоритмов обработки данных реализуются на сервере баз данных и доступны конечному приложению в виде набора *хранимых процедур*, которые и представляют интерфейс обработки данных. Для обеспечения целостности данных, а также в целях безопасности, приложение обычно не получает прямого доступа к данным – вся работа с ними ведется путем вызова тех или иных *хранимых процедур*. Подобный подход делает весьма простой модификацию алгоритмов обработки данных, тотчас же становящихся доступными для всех пользователей сети, и обеспечивает возможность расширения системы без внесения изменений в само приложение: достаточно изменить *хранимую процедуру* на сервере баз данных. Разработчику не нужно перекомпилировать приложение, создавать его копии, а также инструктировать пользователей о необходимости работы с новой версией. Пользователи вообще могут не подозревать о том, что в систему внесены изменения.
Хранимые процедуры существуют независимо от таблиц или каких-либо других объектов баз данных. Они вызываются клиентской программой, другой *хранимой процедурой* или триггером. Разработчик может управлять правами доступа к *хранимой процедуре*, разрешая или запрещая ее *выполнение*. Изменять код *хранимой процедуры* разрешается только ее владельцу или члену фиксированной роли базы данных. При необходимости можно передать права владения ею от одного пользователя к другому.

2 Хранимые процедуры в среде MS SQL Server

При работе с SQL Server пользователи могут создавать собственные процедуры, реализующие те или иные действия. *Хранимые процедуры* являются полноценными объектами базы данных, а потому каждая из них хранится в конкретной базе данных. Непосредственный вызов *хранимой процедуры* возможен, только если он осуществляется в контексте той базы данных, где находится процедура.

Типы хранимых процедур

В SQL Server имеется несколько типов *хранимых процедур*.

- Системные *хранимые процедуры* предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Можно сказать, что системные *хранимые процедуры* являются интерфейсом, обеспечивающим работу с системными таблицами, которая, в конечном счете, сводится к изменению, добавлению, удалению и выборке данных из системных таблиц как пользовательских, так и системных баз данных. Системные *хранимые процедуры* имеют префикс sp_, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.
- Пользовательские *хранимые процедуры* реализуют те или иные действия. *Хранимые процедуры* – полноценный объект базы данных. Вследствие этого каждая *хранимая процедура* располагается в конкретной базе данных, где и выполняется.
- Временные *хранимые процедуры* существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они делятся на локальные и глобальные. Локальные временные *хранимые процедуры* могут быть вызваны только из того соединения, в котором созданы. При *создании* такой процедуры ей необходимо дать имя, начинающееся с одного символа #. Как и все временные объекты, *хранимые процедуры* этого типа автоматически удаляются при отключении пользователя, перезапуске или остановке сервера. Глобальные временные *хранимые процедуры* доступны для любых соединений сервера, на котором имеется такая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

3 Создание, изменение и удаление хранимых процедур

Создание хранимой процедуры предполагает решение следующих задач:

- определение типа создаваемой *хранимой процедуры*: временная или пользовательская. Кроме этого, можно создать свою собственную системную *хранимую процедуру*, назначив ей имя с префиксом sp_ и поместив ее в системную базу данных. Такая процедура будет доступна в контексте любой базы данных локального сервера;
- планирование прав доступа. При *создании хранимой процедуры* следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь;
- определение *параметров хранимой процедуры*. Подобно процедурам, входящим в состав большинства языков программирования, *хранимые процедуры* могут обладать *входными* и выходными *параметрами*;
- разработка кода *хранимой процедуры*. Код процедуры может содержать последовательность любых команд SQL, включая вызов других *хранимых процедур*.

Создание новой и изменение имеющейся *хранимой процедуры* осуществляется с помощью следующей команды:

```
<определение_процедуры>::=
{CREATE | ALTER } [PROCEDURE] имя_процедуры
   [;номер]
[{@имя_параметра тип_данных } [VARYING ]
        [=default][OUTPUT] ][,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE,
        ENCRYPTION }]
[FOR REPLICATION]
AS
        sql_оператор [...n]
```

Рассмотрим *параметры* данной команды.

Используя префиксы sp_, #, ##, создаваемую процедуру можно определить в качестве системной или временной. Как видно из синтаксиса команды, не допускается указывать имя владельца, которому будет принадлежать создаваемая процедура, а также имя базы данных, где она должна быть размещена. Таким образом, чтобы разместить создаваемую *хранимую процедуру* в конкретной базе данных, необходимо выполнить команду CREATE PROCEDURE в контексте этой базы данных. При *обращении* из тела *хранимой процедуры* к объектам той же базы данных можно использовать укороченные имена, т. е. без указания имени базы данных. Когда же требуется обратиться к объектам, расположенным в других базах данных, указание имени базы данных обязательно.

Номер в имени – это идентификационный номер *хранимой процедуры*, однозначно определяющий ее в группе процедур. Для удобства управления процедурами логически однотипные *хранимые процедуры* можно группировать, присваивая им одинаковые имена, но разные идентификационные номера.

Для передачи входных и выходных данных в создаваемой *хранимой процедуре* могут использоваться *параметры*, имена которых, как и имена локальных переменных, должны начинаться с символа @. В одной *хранимой процедуре* можно задать множество *параметров*, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами *параметров* этой процедуры.

Для определения типа данных, который будет иметь соответствующий *параметр хранимой процедуры*, годятся любые типы данных SQL, включая определенные пользователем. Однако тип данных CURSOR может быть использован только как *выходной параметр хранимой процедуры*, т.е. с указанием ключевого слова OUTPUT.

Наличие ключевого слова OUTPUT означает, что соответствующий *параметр* предназначен для возвращения данных из *хранимой процедуры*. Однако это вовсе не означает, что *параметр* не подходит для передачи значений в *хранимую процедуру*. Указание ключевого слова OUTPUT предписывает серверу при выходе из *хранимой процедуры* присвоить текущее значение *параметра* локальной переменной, которая была указана при вызове процедуры в качестве значения *параметра*. Отметим, что при указании ключевого слова OUTPUT значение соответствующего *параметра* при вызове процедуры может быть задано только с помощью локальной переменной. Не разрешается использование любых выражений или констант, допустимое для обычных *параметров*.

Ключевое слово VARYING применяется совместно с *параметром* OUTPUT, имеющим тип CURSOR. Оно определяет, что *выходным параметром* будет результирующее множество.

Ключевое слово DEFAULT представляет собой значение, которое будет принимать соответствующий *параметр по умолчанию*. Таким образом, при вызове процедуры можно не указывать явно значение соответствующего *параметра*.

Так как сервер кэширует *план исполнения* запроса и компилированный код, при последующем вызове процедуры будут использоваться уже готовые значения. Однако в некоторых случаях

все же требуется выполнять перекомпиляцию кода процедуры. Указание ключевого слова RECOMPILE предписывает системе создавать *план выполнения хранимой процедуры* при каждом ее вызове.
Параметр FOR REPLICATION востребован при репликации данных и включении создаваемой *хранимой процедуры* в качестве статьи в публикацию.
Ключевое слово ENCRYPTION предписывает серверу выполнить шифрование кода *хранимой процедуры*, что может обеспечить защиту от использования авторских алгоритмов, реализующих работу *хранимой процедуры*.
Ключевое слово AS размещается в начале собственно тела *хранимой процедуры*, т.е. набора команд SQL, с помощью которых и будет реализовываться то или иное действие. В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие *хранимые процедуры*. Выход из *хранимой процедуры* можно осуществить посредством команды RETURN.
Удаление хранимой процедуры осуществляется командой:

```
DROP PROCEDURE {имя_процедуры} [,...n]
```

4 Выполнение хранимой процедуры

Для *выполнения хранимой процедуры* используется команда:

```
[[ EXEC [ UTE] имя_процедуры [;номер]
[[@имя_параметра=]{значение | @имя_переменной}
        [OUTPUT ]|[DEFAULT ]][,...n]
```

Если вызов *хранимой процедуры* не является единственной командой в пакете, то присутствие команды EXECUTE обязательно. Более того, эта команда требуется для вызова процедуры из тела другой процедуры или триггера.
Использование ключевого слова OUTPUT при вызове процедуры разрешается только для *параметров*, которые были объявлены при *создании процедуры* с ключевым словом OUTPUT. Когда же при вызове процедуры для *параметра* указывается ключевое слово DEFAULT, то будет использовано *значение по умолчанию*. Естественно, указанное слово DEFAULT разрешается только для тех *параметров*, для которых определено *значение по умолчанию*.
Из синтаксиса команды EXECUTE видно, что имена *параметров* могут быть опущены при вызове процедуры. Однако в этом случае пользователь должен указывать значения для *параметров* в том же порядке, в каком они перечислялись при *создании процедуры*. Присвоить *параметру значение по умолчанию*, просто пропустив его при перечислении нельзя. Если же требуется опустить *параметры*, для которых определено *значение по умолчанию*, достаточно явного указания имен *параметров* при вызове *хранимой процедуры*. Более того, таким способом можно перечислять *параметры* и их значения в произвольном порядке.
Отметим, что при вызове процедуры указываются либо имена *параметров* со значениями, либо только значения без имени *параметра*. Их комбинирование не допускается.
Пример 12.1. *Процедура без параметров*. Разработать процедуру для получения названий и стоимости товаров, приобретенных Ивановым.

```
CREATE PROC my_proc1
AS
SELECT Товар.Название,
      Товар.Цена*Сделка.Количество
          AS Стоимость, Клиент.Фамилия
  FROM Клиент INNER JOIN
  (Товар INNER JOIN Сделка
  ON Товар.КодТовара=Сделка.КодТовара)
  ON Клиент.КодКлиента=Сделка.КодКлиента
  WHERE Клиент.Фамилия='Иванов'
```

Пример 12.1. Процедура для получения названий и стоимости товаров, приобретенных Ивановым. (html, txt)

Для *обращения к процедуре* можно использовать команды:

```
EXEC my_proc1 или my_proc1
```

Процедура возвращает набор данных.

Пример 12.2. *Процедура без параметров*. Создать процедуру для уменьшения цены товара первого сорта на 10%.

```
CREATE PROC my_proc2
AS
UPDATE Товар SET Цена=Цена*0.9
  WHERE Сорт='первый'
```

Пример 12.2. Процедура для уменьшения цены товара первого сорта на 10%. (html, txt)

Для *обращения к процедуре* можно использовать команды:

```
EXEC my_proc2 или my_proc2
```

Процедура не возвращает никаких данных.

Пример 12.3. *Процедура с входным параметром*. Создать процедуру для получения названий и стоимости товаров, которые приобрел заданный клиент.

```
CREATE PROC my_proc3
  @k VARCHAR(20)
AS
SELECT Товар.Название,
      Товар.Цена*Сделка.Количество
  AS Стоимость, Клиент.Фамилия
  FROM Клиент INNER JOIN
  (Товар INNER JOIN Сделка
  ON Товар.КодТовара=Сделка.КодТовара)
  ON Клиент.КодКлиента=Сделка.КодКлиента
  WHERE Клиент.Фамилия=@k
```

Пример 12.3. Процедура для получения названий и стоимости товаров, которые приобрел заданный клиент. (html, txt)

Для *обращения к процедуре* можно использовать команды:

```
EXEC my_proc3 'Иванов' или
my_proc3 @k='Иванов'
```

Пример 12.4. *Процедура с входными параметрами*. Создать процедуру для уменьшения цены товара заданного типа в соответствии с указанным %.

```
CREATE PROC my_proc4
  @t VARCHAR(20), @p FLOAT
AS
UPDATE Товар SET Цена=Цена*(1-@p)
  WHERE Тип=@t
```

Пример 12.4. Процедура для уменьшения цены товара заданного типа в соответствии с указанным %. (html, txt)

Для *обращения к процедуре* можно использовать команды:

```
EXEC my_proc4 'Вафли',0.05 или
EXEC my_proc4 @t='Вафли', @p=0.05
```

Пример 12.5. *Процедура с входными параметрами* и значениями по умолчанию. Создать процедуру для уменьшения цены товара заданного типа в соответствии с указанным %.

```
CREATE PROC my_proc5
  @t VARCHAR(20)='Конфеты',
  @p FLOAT=0.1
AS
UPDATE Товар SET Цена=Цена*(1-@p)
  WHERE Тип=@t
```

Пример 12.5. Процедура с входными параметрами и значениями по умолчанию. Создать процедуру для уменьшения цены товара заданного типа в соответствии с указанным %. (html, txt)

Для *обращения к процедуре* можно использовать команды:

```
EXEC my_proc5 'Вафли',0.05 или
EXEC my_proc5 @t='Вафли', @p=0.05 или
EXEC my_proc5 @p=0.05
```

В этом случае уменьшается цена конфет (значение типа не указано при вызове процедуры и берется по умолчанию).

```
EXEC my_proc5
```

В последнем случае оба *параметра* (и тип, и проценты) не указаны при вызове процедуры, их значения берутся по умолчанию.

Пример 12.6. *Процедура с входными и выходными параметрами*. Создать процедуру для определения общей стоимости товаров, проданных за конкретный месяц.

```
CREATE PROC my_proc6
  @m INT,
  @s FLOAT OUTPUT
AS
SELECT @s=Sum(Товар.Цена*Сделка.Количество)
  FROM Товар INNER JOIN Сделка
  ON Товар.КодТовара=Сделка.КодТовара
  GROUP BY Month(Сделка.Дата)
  HAVING Month(Сделка.Дата)=@m
```

Пример 12.6. Процедура с входными и выходными параметрами. Создать процедуру для определения общей стоимости товаров, проданных за конкретный месяц. (html, txt)

Для *обращения к процедуре* можно использовать команды:

```
DECLARE @st FLOAT
EXEC my_proc6 1,@st OUTPUT
SELECT @st
```

Этот блок команд позволяет определить стоимость товаров, проданных в январе (*входной параметр* месяц указан равным 1).

Создать процедуру для определения общего количества товаров, приобретенных фирмой, в которой работает заданный сотрудник.

Сначала разработаем процедуру для определения фирмы, где работает сотрудник.

```
CREATE PROC my_proc7
  @n VARCHAR(20),
  @f VARCHAR(20) OUTPUT
AS
SELECT @f=Фирма
  FROM Клиент
  WHERE Фамилия=@n
```

Пример 12.7. Использование *вложенных процедур*. Создать процедуру для определения общего количества товаров, приобретенных фирмой, в которой работает заданный сотрудник.

Затем создадим процедуру, подсчитывающую общее количество товара, который закуплен интересующей нас фирмой.

```
CREATE PROC my_proc8
  @fam VARCHAR(20),
  @kol INT OUTPUT
AS
DECLARE @firm VARCHAR(20)
EXEC my_proc7 @fam,@firm OUTPUT
SELECT @kol=Sum(Сделка.Количество)
  FROM Клиент INNER JOIN Сделка
  ON Клиент.КодКлиента=Сделка.КодКлиента
  GROUP BY Клиент.Фирма
  HAVING Клиент.Фирма=@firm
```

Пример 12.7. Создание процедуры для определения общего количества товаров, приобретенных фирмой, в которой работает заданный сотрудник. (html, txt)

Вызов процедуры осуществляется с помощью команды:

```
DECLARE @k INT
EXEC my_proc8 ‘Иванов’,@k OUTPUT
SELECT @k
```

Лекция: Курсоры: принципы работы

1 Понятие курсора

Запрос к реляционной базе данных обычно возвращает несколько рядов (записей) данных, но приложение за один раз обрабатывает лишь одну запись. Даже если оно имеет дело одновременно с несколькими рядами (например, выводит данные в форме электронных таблиц), их количество по-прежнему ограничено. Кроме того, при модификации, удалении или добавлении данных рабочей единицей является ряд. В этой ситуации на первый план выступает концепция *курсора*, и в таком контексте курсор – указатель на ряд.
Курсор в SQL – это область в памяти базы данных, которая предназначена для хранения последнего оператора SQL. Если текущий оператор – запрос к базе данных, в памяти сохраняется и строка данных запроса, называемая текущим значением, или текущей строкой *курсора*. Указанная область в памяти поименована и доступна для прикладных программ. Обычно *курсоры* используются для выбора из базы данных некоторого подмножества хранимой в ней информации. В каждый момент времени прикладной программой может быть проверена одна строка *курсора*. *Курсоры* часто применяются в операторах SQL, встроенных в написанные на языках процедурного типа прикладные программы. Некоторые из них неявно создаются сервером базы данных, в то время как другие определяются программистами.
В соответствии со стандартом SQL при работе с *курсорами* можно выделить следующие основные *действия*:

- создание или *объявление курсора*;
- ***открытие курсора*** , т.е. наполнение его данными, которые сохраняются в многоуровневой памяти;
- *выборка из курсора* и *изменение* с его помощью строк данных;
- *закрытие курсора*, после чего он становится недоступным для пользовательских программ;
- *освобождение курсора*, т.е. удаление *курсора* как объекта, поскольку его *закрытие* необязательно освобождает ассоциированную с ним память.

В разных реализациях определение *курсора* может иметь некоторые отличия. Так, например, иногда разработчик должен явным образом освободить выделяемую для *курсора* память. После *освобождения курсора* ассоциированная с ним память также освобождается. При этом становится возможным повторное использование его имени. В других реализациях при *закрытии курсора* освобождение памяти происходит неявным образом. Сразу после восстановления она становится доступной для других операций: *открытие другого курсора* и т.д.
В некоторых случаях применение *курсора* неизбежно. Однако по возможности этого следует избегать и работать со стандартными командами обработки данных: SELECT, UPDATE, INSERT, DELETE. Помимо того, что *курсоры* не позволяют проводить операции *изменения* над всем объемом данных, скорость выполнения операций обработки данных посредством *курсора* заметно ниже, чем у стандартных средств SQL.

2 Реализация курсоров в среде MS SQL Server

SQL Server поддерживает три *вида курсоров*:

- *курсоры* SQL применяются в основном внутри триггеров, хранимых процедур и сценариев;
- *курсоры* сервера действуют на сервере и реализуют программный интерфейс приложений для ODBC, OLE DB, DB_Library;
- *курсоры* клиента реализуются на самом клиенте. Они выбирают весь результирующий набор строк из сервера и сохраняют его локально, что позволяет ускорить операции обработки данных за счет снижения потерь времени на выполнение сетевых операций.

Различные типы многопользовательских приложений требуют и различных типов организации параллельного доступа к данным. Некоторым приложениям необходим немедленный доступ к информации об *изменениях* в базе данных. Это характерно для систем резервирования билетов. В других случаях, например, в системах статистической отчетности, важна стабильность данных, ведь если они постоянно модифицируются, программы не смогут эффективно отображать информацию. Различным приложениям нужны разные реализации *курсоров*.
В среде SQL Server типы *курсоров* различаются по предоставляемым возможностям. Тип *курсора* определяется на стадии его создания и не может быть изменен. Некоторые типы *курсоров* могут обнаруживать *изменения*, сделанные другими пользователями в строках, включенных в результирующий набор. Однако SQL Server отслеживает *изменения* таких строк только на стадии обращения к строке и не позволяет отслеживать изменения, когда строка уже считана.
Курсоры делятся на две категории: *последовательные* и *прокручиваемые*. ***Последовательные*** позволяют выбирать данные только в одном направлении – от начала к концу. *Прокручиваемые же курсоры* предоставляют большую свободу действий – допускается перемещение в обоих направлениях и переход к произвольной строке результирующего набора *курсора*.Если программа способна модифицировать данные, на которые указывает *курсор*, он называется ***прокручиваемым*** и модифицируемым. Говоря о *курсорах*, не следует забывать об изолированности транзакций. Когда один пользователь модифицирует запись, другой читает ее при помощи собственного *курсора*, более того, он может модифицировать ту же запись, что делает необходимым соблюдение целостности данных.
SQL Server поддерживает *курсоры статические*, *динамические*, *последовательные* и управляемые набором ключей.
В схеме со ***статическим курсором*** информация читается из базы данных один раз и хранится в виде моментального снимка (по состоянию на некоторый момент времени), поэтому изменения, внесенные в базу данных другим пользователем, не видны. На время *открытия курсора* сервер устанавливает блокировку на все строки, включенные в его полный результирующий набор. *Статический курсор* не изменяется после создания и всегда отображает тот набор данных, который существовал на момент его *открытия*.
Если другие пользователи изменят в исходной таблице включенные в *курсор* данные, это никак не повлияет на *статический курсор*.
В *статический курсор* внести *изменения* невозможно, поэтому он всегда открывается в режиме "только для чтения".
Динамический курсор поддерживает данные в "живом" состоянии, но это требует затрат сетевых и программных ресурсов. При использовании *динамических курсоров* не создается полная копия исходных данных, а выполняется динамическая выборка из исходных таблиц только при обращении пользователя к тем или иным данным. На время выборки сервер блокирует строки, а все *изменения*, вносимые пользователем в полный результирующий набор *курсора*, будут видны в *курсоре*. Однако если другой пользователь внес *изменения* уже после выборки данных *курсором*, то они не отразятся в *курсоре*.
Курсор, управляемый набором ключей, находится посередине между этими крайностями. Записи идентифицируются на момент выборки, и тем самым отслеживаются *изменения*. Такой тип *курсора* полезен при реализации прокрутки назад – тогда добавления и *удаления рядов* не

видны, пока информация не обновится, а драйвер выбирает новую версию записи, если в нее были внесены *изменения*.
Последовательные курсоры не разрешают выполнять выборку данных в обратном направлении. Пользователь может выбирать строки только от начала к концу *курсора*. *Последовательный курсор* не хранит набор всех строк. Они считываются из базы данных, как только выбираются в *курсоре*, что позволяет динамически отражать все *изменения*, вносимые пользователями в базу данных с помощью команд INSERT, UPDATE, DELETE. В *курсоре* видно самое последнее состояние данных.
Статические курсоры обеспечивают стабильный взгляд на данные. Они хороши для систем "складирования" информации: приложений для систем отчетности или для статистических и аналитических целей. Кроме того, *статический курсор* лучше других справляется с выборкой большого количества данных. Напротив, в системах электронных покупок или резервирования билетов необходимо динамическое восприятие обновляемой информации по мере внесения изменений. В таких случаях используется *динамический курсор*. В этих приложениях объем передаваемых данных, как правило, невелик, а доступ к ним осуществляется на уровне рядов (отдельных записей). Групповой доступ встречается очень редко.

3 Управление курсором в среде MS SQL Server

Управление курсором реализуется путем выполнения следующих команд:

- DECLARE – создание или *объявление курсора*;
- OPEN – *открытие курсора*, т.е. наполнение его данными;
- FETCH – *выборка из курсора* и *изменение* строк данных с помощью курсора;
- CLOSE – *закрытие курсора*;
- DEALLOCATE – *освобождение курсора*, т.е. удаление курсора как объекта.

Объявление курсора

В стандарте SQL для создания *курсора* предусмотрена следующая команда:

```
<создание_курсора>::=
  DECLARE имя_курсора
    [INSENSITIVE][SCROLL] CURSOR
  FOR SELECT_оператор
  [FOR { READ_ONLY | UPDATE
    [OF имя_столбца[,...n]]}]
```

При использовании ключевого слова INSENSITIVE будет создан *статический курсор*. *Изменения данных* не разрешаются, кроме того, не отображаются *изменения*, сделанные другими пользователями. Если ключевое слово INSENSITIVE отсутствует, создается *динамический курсор*.
При указании ключевого слова SCROLL созданный *курсор* можно прокручивать в любом направлении, что позволяет применять любые команды *выборки*. Если этот аргумент опускается, то *курсор* окажется *последовательным*, т.е. его просмотр будет возможен только в одном направлении – от начала к концу.
SELECT-оператор задает тело запроса SELECT, с помощью которого определяется результирующий набор строк *курсора*.
При указании аргумента FOR READ_ONLY создается *курсор* "только для чтения", и никакие модификации данных не разрешаются. Он отличается от *статического*, хотя последний также не позволяет менять данные. В качестве курсора "только для чтения" может быть объявлен *динамический курсор*, что позволит отображать *изменения*, сделанные другим пользователем.

Создание *курсора* с аргументом FOR UPDATE позволяет выполнять в *курсоре изменение данных* либо в указанных столбцах, либо, при отсутствии аргумента OF имя_столбца, во всех столбцах.
В среде MS SQL Server принят следующий синтаксис команды создания *курсора*:

```
<создание_курсора>::=
  DECLARE имя_курсора CURSOR [LOCAL | GLOBAL]
  [FORWARD_ONLY | SCROLL]
  [STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
  [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
  [TYPE_WARNING]
  FOR SELECT_оператор
  [FOR UPDATE [OF имя_столбца[,...n]]]
```

При использовании ключевого слова LOCAL будет создан локальный *курсор*, который виден только в пределах создавшего его пакета, триггера, хранимой процедуры или пользовательской функции. По завершении работы пакета, триггера, процедуры или функции *курсор* неявно уничтожается. Чтобы передать содержимое *курсора* за пределы создавшей его конструкции, необходимо присвоить его параметру аргумент OUTPUT.
Если указано ключевое слово GLOBAL, создается глобальный *курсор*; он существует до закрытия текущего соединения.
При указании FORWARD_ONLY создается *последовательный курсор*; *выборку* данных можно осуществлять только в направлении от первой строки к последней.
При указании SCROLL создается *прокручиваемый курсор*; обращаться к данным можно в любом порядке и в любом направлении.
При указании STATIC создается *статический курсор*.
При указании KEYSET создается ключевой курсор.
При указании DYNAMIC создается *динамический курсор*.
Если для *курсора* READ_ONLY указать аргумент FAST_FORWARD, то созданный *курсор* будет оптимизирован для быстрого доступа к данным. Этот аргумент не может быть использован совместно с аргументами FORWARD_ONLY и OPTIMISTIC.
В *курсоре*, созданном с указанием аргумента OPTIMISTIC, запрещается *изменение* и *удаление строк*, которые были изменены после *открытия курсора*.
При указании аргумента TYPE_WARNING сервер будет информировать пользователя о неявном изменении типа *курсора*, если он несовместим с запросом SELECT.

Открытие курсора

Для *открытия курсора* и наполнения его данными из указанного при создании *курсора* запроса SELECT используется следующая команда:

```
OPEN {{[GLOBAL]имя_курсора }
    |@имя_переменной_курсора}
```

После *открытия курсора* происходит выполнение связанного с ним оператора SELECT, выходные данные которого сохраняются в многоуровневой памяти.

Выборка данных из курсора

Сразу после *открытия курсора* можно выбрать его содержимое (результат выполнения соответствующего запроса) посредством следующей команды:

```
FETCH [[NEXT | PRIOR | FIRST | LAST
  | ABSOLUTE {номер_строки
  | @переменная_номера_строки}
  | RELATIVE {номер_строки |
    @переменная_номера_строки}]
  FROM ]{{[GLOBAL ]имя_курсора }|
```

@имя_переменной_курсора }
[INTO @имя_переменной [,...n]]

При указании FIRST будет возвращена самая первая строка полного результирующего набора *курсора*, которая становится текущей строкой.

При указании LAST возвращается самая последняя строка *курсора*. Она же становится текущей строкой.

При указании NEXT возвращается строка, находящаяся в полном результирующем наборе сразу же после текущей. Теперь она становится текущей. По умолчанию команда FETCH использует именно этот способ *выборки* строк.

Ключевое слово PRIOR возвращает строку, находящуюся перед текущей. Она и становится текущей.

Аргумент ABSOLUTE {номер_строки | @переменная_номера_строки} возвращает строку по ее абсолютному порядковому номеру в полном результирующем наборе *курсора*. Номер строки можно задать с помощью константы или как имя переменной, в которой хранится номер строки. Переменная должна иметь целочисленный тип данных. Указываются как положительные, так и отрицательные значения. При указании положительного значения строка отсчитывается от начала набора, отрицательного – от конца. Выбранная строка становится текущей. Если указано нулевое значение, строка не возвращается.

Аргумент RELATIVE {кол_строки | @переменная_кол_строки} возвращает строку, находящуюся через указанное количество строк после текущей. Если указать отрицательное значение числа строк, то будет возвращена строка, находящаяся за указанное количество строк перед текущей. При указании нулевого значения возвратится текущая строка. Возвращенная строка становится текущей.

Чтобы *открыть глобальный курсор*, перед его именем требуется указать ключевое слово GLOBAL. Имя *курсора* также может быть указано с помощью переменной.

В конструкции INTO @имя_переменной [,...n] задается список переменных, в которых будут сохранены соответствующие значения столбцов возвращаемой строки. Порядок указания переменных должен соответствовать порядку столбцов в *курсоре*, а тип данных переменной – типу данных в столбце *курсора*. Если конструкция INTO не указана, то поведение команды FETCH будет напоминать поведение команды SELECT – данные выводятся на экран.

Изменение и удаление данных

Для выполнения изменений с помощью *курсора* необходимо выполнить команду UPDATE в следующем формате:

```
UPDATE имя_таблицы SET {имя_столбца={
  DEFAULT | NULL | выражение}}[,...n]
  WHERE CURRENT OF {{[GLOBAL] имя_курсора}
  |@имя_переменной_курсора}
```

За одну операцию могут быть изменены несколько столбцов текущей строки *курсора*, но все они должны принадлежать одной таблице.

Для удаления данных посредством *курсора* используется команда DELETE в следующем формате:

```
DELETE имя_таблицы
  WHERE CURRENT OF {{[GLOBAL] имя_курсора}
  |@имя_переменной_курсора}
```

В результате будет удалена строка, установленная текущей в *курсоре*.

Закрытие курсора

```
CLOSE {имя_курсора | @имя_переменной_курсора}
```

После *закрытия курсор* становится недоступным для пользователей программы. При *закрытии* снимаются все блокировки, установленные в процессе его работы. *Закрытие* может

применяться только к открытым *курсорам*. Закрытый, но не *освобожденный курсор* может быть повторно открыт. Не допускается закрывать неоткрытый *курсор*.

Освобождение курсора

Закрытие курсора необязательно освобождает ассоциированную с ним память. В некоторых реализациях нужно явным образом освободить ее с помощью оператора DEALLOCATE. После *освобождения курсора* освобождается и память, при этом становится возможным повторное использование имени *курсора*.

```
DEALLOCATE { имя_курсора |
  @имя_переменной_курсора }
```

Для контроля достижения конца *курсора* рекомендуется применять функцию:
@@FETCH_STATUS
Функция @@FETCH_STATUS возвращает:
0, если *выборка* завершилась успешно;
-1, если *выборка* завершилась неудачно вследствие попытки *выборки* строки, находящейся за пределами *курсора*;
-2, если *выборка* завершилась неудачно вследствие попытки обращения к удаленной или измененной строке.

```
DECLARE abc CURSOR SCROLL FOR
SELECT * FROM Клиент
```

Пример 13.1. Объявление курсора. (html, txt)

```
DECLARE @MyCursor CURSOR
SET @MyCursor=CURSOR LOCAL SCROLL FOR
SELECT * FROM Клиент
```

Пример 13.2. Использование переменной для объявления курсора. (html, txt)

```
DECLARE abc CURSOR GLOBAL SCROLL FOR
SELECT * FROM Клиент
OPEN abc
```

Пример 13.3. Объявление и открытие курсора. (html, txt)

```
DECLARE @MyCursor CURSOR
SET @MyCursor=abc
```

Пример 13.4. Использование переменной для переприсваивания курсора. (html, txt)

Пример 13.5. Разработать курсор для вывода списка фирм и клиентов из Москвы.

```
DECLARE  @firm    VARCHAR(50),
         @fam     VARCHAR(50),
         @message VARCHAR(80)
PRINT ' Список клиентов'
DECLARE klient_cursor CURSOR LOCAL FOR
    SELECT Фирма, Фамилия
    FROM Клиент
    WHERE Город='Москва'
    ORDER BY Фирма, Фамилия

OPEN klient_cursor
FETCH NEXT FROM klient_cursor INTO @firm, @fam
WHILE @@FETCH_STATUS=0
BEGIN
```

```
    SELECT @message='Клиент '+@fam+
                ' Фирма '+ @firm
    PRINT @message

  -- переход к следующему клиенту--

    FETCH NEXT FROM klient_cursor
      INTO @firm, @fam
  END
  CLOSE klient_cursor
  DEALLOCATE klient_cursor
```

Пример 13.5. Курсор для вывода списка фирм и клиентов из Москвы. (html, txt)

Пример 13.6. Разработать курсор для вывода списка приобретенных клиентами из Москвы товаров и их общей стоимости. В один курсор заносятся все московские клиенты, затем для каждой строки курсора, т.е. для каждого клиента, определяется и распечатывается другой курсор – его покупки. Подсчитывается общая стоимость покупок клиента.

```
  DECLARE @id_kl    INT,
          @firm     VARCHAR(50),
          @fam      VARCHAR(50),
          @message  VARCHAR(80),
          @nam      VARCHAR(50),
          @d        DATETIME,
          @p        INT,
          @s        INT
  SET @s=0
  PRINT '  Список покупок'
  DECLARE klient_cursor CURSOR LOCAL FOR
    SELECT КодКлиента, Фирма, Фамилия
    FROM Клиент
    WHERE Город='Москва'
    ORDER BY Фирма, Фамилия


  OPEN klient_cursor
  FETCH NEXT FROM klient_cursor
  INTO @id_kl, @firm, @fam
  WHILE @@FETCH_STATUS=0
  BEGIN
    SELECT @message='Клиент '+@fam+
     ' Фирма '+ @firm
    PRINT @message
    SELECT @message='Наименование товара Дата
      покупки Стоимость'
    PRINT @message
    DECLARE tovar_cursor CURSOR FOR
      SELECT Товар.Название, Сделка.Дата,
        Товар.Цена*Сделка.Количество AS
    Стоимость
      FROM Товар INNER JOIN Сделка ON Товар.
    КодТовара=Сделка.КодТовара
```

```
        WHERE Сделка.КодКлиента=@id_kl

    OPEN tovar_cursor
    FETCH NEXT FROM tovar_cursor
      INTO @nam, @d, @p
    IF @@FETCH_STATUS<>0
       PRINT ' Нет покупок'
    WHILE @@FETCH_STATUS=0
    BEGIN
       SELECT @message='   '+@nam+'   '+
         CAST(@d AS CHAR(12))+'  '+
         CAST(@p AS CHAR(6))
       PRINT @message
       SET @s=@s+@p
       FETCH NEXT FROM tovar_cursor
       INTO @nam, @d, @p
    END
    CLOSE tovar_cursor
    DEALLOCATE tovar_cursor

    SELECT @message='Общая стоимость '+
              CAST(@s AS CHAR(6))
    PRINT @message

  -- переход к следующему клиенту--

    FETCH NEXT FROM klient_cursor
    INTO @id_kl, @firm, @fam
  END
  CLOSE klient_cursor
  DEALLOCATE klient_cursor
```

Пример 13.6. Курсор для вывода списка приобретенных клиентами из Москвы товаров и их общей стоимости. (html, txt)

Пример 13.7. Разработать прокручиваемый курсор для клиентов из Москвы. Если номер телефона начинается на 1, удалить клиента с таким номером и в первой записи курсора заменить первую цифру в номере телефона на 4.

```
  DECLARE @firm    VARCHAR(50),
          @fam     VARCHAR(50),
          @tel     VARCHAR(8),
          @message  VARCHAR(80)
  PRINT '  Список клиентов'
  DECLARE klient_cursor CURSOR GLOBAL SCROLL
  KEYSET FOR
          SELECT Фирма, Фамилия, Телефон
          FROM Клиент
          WHERE Город='Москва'
          ORDER BY Фирма, Фамилия
  FOR UPDATE
  OPEN klient_cursor
  FETCH NEXT FROM klient_cursor
```

```
   INTO  @firm, @fam, @tel
WHILE @@FETCH_STATUS=0
BEGIN
   SELECT @message='Клиент '+@fam+
      '  Фирма  '+@firm '  Телефон '+ @tel
   PRINT @message

-- если номер телефона начинается на 1,
-- удалить клиента с таким номером
   IF @tel LIKE '1%'
     DELETE Клиент
     WHERE CURRENT OF klient_cursor
   ELSE

-- переход к следующему клиенту

   FETCH NEXT FROM klient_cursor
          INTO @firm, @fam, @tel
END

FETCH ABSOLUTE 1 FROM klient_cursor
   INTO @firm, @fam, @tel

-- в первой записи заменить первую цифру в
-- номере телефона на 4

  UPDATE Клиент SET Телефон='4' +
  RIGHT(@tel,LEN(@tel)-1))
      WHERE CURRENT OF klient_cursor
SELECT @message='Клиент '+@fam+'  Фирма  '+
   @firm '  Телефон '+ @tel
   PRINT @message
CLOSE klient_cursor
DEALLOCATE klient_cursor
```

Пример 13.7. Прокручиваемый курсор для клиентов из Москвы. (html, txt)

Пример 13.8. Использование *курсора как выходного параметра процедуры*. Процедура возвращает набор данных – список товаров.

```
CREATE PROC my_proc
@cur CURSOR VARYING OUTPUT
AS
SET @cur=CURSOR FORWARD_ONLY STATIC FOR
SELECT Название FROM Товар
OPEN @cur
```

Пример 13.8. Использование курсора как выходного параметра процедуры. (html, txt)

Вызов процедуры и вывод на печать данных из выходного *курсора* осуществляется следующим образом:

```
DECLARE @my_cur CURSOR
DECLARE @n VARCHAR(20)
EXEC my_proc @cur=@my_cur OUTPUT
  FETCH NEXT FROM @my_cur INTO @n
```

```
  SELECT @n
WHILE (@@FETCH_STATUS=0)
BEGIN
  FETCH NEXT FROM @my_cur INTO @n
  SELECT @n
END
CLOSE @my_cur
DEALLOCATE @my_cur
```

Лекция: Триггеры: создание и применение

1 Определение триггера в стандарте языка SQL

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). *Триггеры* используются для проверки целостности данных, а также для отката транзакций.
Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение *триггеров* большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение *триггеров* нецелесообразно.
Триггеры – особый инструмент SQL-сервера, используемый для поддержания целостности данных в базе данных. С помощью ограничений целостности, правил и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения значений таблицы, чтобы нужным образом изменить связанные данные. *Триггеры* можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т.д.
Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми *триггеры* связаны. Каждый *триггер* привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные *триггером*.
Создает *триггер* только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.
Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому *триггеры* необходимо очень тщательно отлаживать.
В отличие от обычной подпрограммы, *триггер* выполняется неявно в каждом случае возникновения *триггерного события*, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском *триггера*. С помощью *триггеров* достигаются следующие цели:

- проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
- выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
- накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
- поддержка репликации.

Основной формат команды CREATE TRIGGER показан ниже:
<Определение_триггера>::=

```
CREATE TRIGGER имя_триггера
BEFORE | AFTER <триггерное_событие>
ON <имя_таблицы>
[REFERENCING
  <список_старых_или_новых_псевдонимов>]
[FOR EACH { ROW | STATEMENT}]
[WHEN(условие_триггера)]
<тело_триггера>
```

триггерные события состоят из вставки, удаления и обновления строк в таблице. В последнем случае для *триггерного события* можно указать конкретные имена столбцов таблицы. Время запуска *триггера* определяется с помощью ключевых слов BEFORE (*триггер* запускается до выполнения связанных с ним событий) или AFTER (после их выполнения).
Выполняемые *триггером* действия задаются для каждой строки (FOR EACH ROW), охваченной данным событием, или только один раз для каждого события (FOR EACH STATEMENT). Обозначение <список_старых_или_новых_псевдонимов> относится к таким компонентам, как старая или новая строка (OLD / NEW) либо старая или новая таблица (OLD TABLE / NEW TABLE). Ясно, что старые значения не применимы для событий вставки, а новые – для событий удаления.
При условии правильного использования *триггеры* могут стать очень мощным механизмом. Основное их преимущество заключается в том, что стандартные функции сохраняются внутри базы данных и согласованно активизируются при каждом ее обновлении. Это может существенно упростить приложения. Тем не менее следует упомянуть и о присущих *триггеру* недостатках:

- сложность: при перемещении некоторых функций в базу данных усложняются задачи ее проектирования, реализации и администрирования;
- скрытая функциональность: перенос части функций в базу данных и сохранение их в виде одного или нескольких *триггеров* иногда приводит к сокрытию от пользователя некоторых функциональных возможностей. Хотя это в определенной степени упрощает его работу, но, к сожалению, может стать причиной незапланированных, потенциально нежелательных и вредных побочных эффектов, поскольку в этом случае пользователь не в состоянии контролировать все процессы, происходящие в базе данных;
- влияние на производительность: перед выполнением каждой команды по изменению состояния базы данных СУБД должна проверить триггерное условие с целью выяснения необходимости запуска *триггера* для этой команды. Выполнение подобных вычислений сказывается на общей производительности СУБД, а в моменты пиковой нагрузки ее снижение может стать особенно заметным. Очевидно, что при возрастании количества *триггеров* увеличиваются и накладные расходы, связанные с такими операциями.

Неправильно написанные *триггеры* могут привести к серьезным проблемам, таким, например, как появление "мертвых" блокировок. *Триггеры* способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

2 Реализация триггеров в среде MS SQL Server

В реализации СУБД MS SQL Server используется следующий оператор создания или изменения *триггера*:

```
<Определение_триггера>::=
{CREATE | ALTER} TRIGGER имя_триггера
ON {имя_таблицы | имя_просмотра }
```

```
[WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF }
{ [ DELETE] [,] [ INSERT] [,] [ UPDATE] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
  sql_оператор[...n]
} |
{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
 [UPDATE] }
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS
{ IF UPDATE(имя_столбца)
[ {AND | OR} UPDATE(имя_столбца)] [...n]
|
IF (COLUMNS_UPDATES(){оператор_бит_обработки}
 бит_маска_изменения)
{оператор_бит_сравнения }бит_маска [...n]}
sql_оператор [...n]
}
}
```

Триггер может быть создан только в текущей базе данных, но допускается обращение внутри *триггера* к другим базам данных, в том числе и расположенным на удаленном сервере.
Рассмотрим назначение аргументов из команды CREATE | ALTER TRIGGER.
Имя *триггера* должно быть уникальным в пределах базы данных. Дополнительно можно указать имя владельца.
При указании аргумента WITH ENCRYPTION сервер выполняет шифрование кода *триггера*, чтобы никто, включая администратора, не мог получить к нему доступ и прочитать его. Шифрование часто используется для скрытия авторских алгоритмов обработки данных, являющихся интеллектуальной собственностью программиста или коммерческой тайной.

Типы триггеров

В SQL Server существует два параметра, определяющих поведение *триггеров*:

- AFTER. *Триггер* выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, *триггер* не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение *триггера* осуществляется в теле одной транзакции: если произойдет откат *триггера*, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER-триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все *триггеры* являются AFTER-триггерами.
- INSTEAD OF. *Триггер* вызывается вместо выполнения команд. В отличие от AFTER-триггера INSTEAD OF-триггер может быть определен как для таблицы, так и для просмотра. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF-триггер.

Триггеры различают по типу команд, на которые они реагируют.
Существует три типа *триггеров*:

- INSERT TRIGGER – запускаются при попытке вставки данных с помощью команды INSERT.
- UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды UPDATE.
- DELETE TRIGGER – запускаются при попытке удаления данных с помощью команды DELETE.

Конструкции [DELETE] [,] [INSERT] [,] [UPDATE] и FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] определяют, на какую команду будет реагировать *триггер*. При его создании должна быть указана хотя бы одна команда. Допускается *создание триггера*, реагирующего на две или на все три команды.
Аргумент WITH APPEND позволяет создавать несколько *триггеров* каждого типа.
При *создании триггера* с аргументом NOT FOR REPLICATION запрещается его запуск во время выполнения модификации таблиц механизмами репликации.
Конструкция AS sql_оператор[...n] определяет набор SQL- операторов и команд, которые будут выполнены при запуске *триггера*.
Отметим, что внутри *триггера* не допускается выполнение ряда операций, таких, например, как:

- создание, изменение и удаление базы данных;
- восстановление резервной копии базы данных или журнала транзакций.

Выполнение этих команд не разрешено, так как они не могут быть отменены в случае отката транзакции, в которой выполняется *триггер*. Это запрещение вряд ли может каким-то образом сказаться на функциональности создаваемых *триггеров*. Трудно найти такую ситуацию, когда, например, после изменения строки таблицы потребуется выполнить восстановление резервной копии журнала транзакций.

3 Программирование триггера

При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: *inserted* и *deleted*. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется *триггер*. Для каждого *триггера* создается свой комплект таблиц inserted и deleted, поэтому никакой другой *триггер* не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение *триггера*, содержимое таблиц inserted и deleted может быть разным:

- команда INSERT – в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни одной строки; после завершения *триггера* все строки из таблицы inserted переместятся в исходную таблицу;
- команда DELETE – в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; *триггер* может проверить каждую строку и определить, разрешено ли ее удаление; в таблице inserted не окажется ни одной строки;
- команда UPDATE – при ее выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении *триггера*. Новые значения строк содержатся в таблице inserted. Эти строки добавятся в исходную таблицу после успешного выполнения *триггера*.

Для получения информации о количестве строк, которое будет изменено при успешном завершении *триггера*, можно использовать функцию @@ROWCOUNT; она возвращает количество строк, обработанных последней командой. Следует подчеркнуть, что *триггер* запускается не при попытке изменить конкретную строку, а в момент выполнения команды изменения. Одна такая команда воздействует на множество строк, поэтому *триггер* должен обрабатывать все эти строки.
Если *триггер* обнаружил, что из 100 вставляемых, изменяемых или удаляемых строк только одна не удовлетворяет тем или иным условиям, то никакая строка не будет вставлена, изменена или удалена. Такое поведение обусловлено требованиями транзакции – должны быть выполнены либо все модификации, либо ни одной.
Триггер выполняется как неявно определенная транзакция, поэтому внутри *триггера* допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения *триггера* и отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду ROLLBACK TRANSACTION.
Для получения списка столбцов, измененных при выполнении команд INSERT или UPDATE, вызвавших выполнение *триггера*, можно использовать функцию COLUMNS_UPDATED(). Она возвращает двоичное число, каждый бит которого, начиная с младшего, соответствует одному столбцу таблицы (в порядке следования столбцов при создании таблицы). Если бит установлен в значение "1", то соответствующий столбец был изменен. Кроме того, факт изменения столбца определяет и функция UPDATE (имя_столбца).
Для *удаления триггера* используется команда

```
DROP TRIGGER {имя_триггера} [,...n]
```

Приведем примеры использования *триггеров*.

Пример 14.1. Использование *триггера* для *реализации ограничений на значение*. В добавляемой в таблицу Сделка записи количество проданного товара должно быть не больше, чем его остаток из таблицы Склад.

Команда вставки записи в таблицу Сделка может быть, например, такой:

```
INSERT INTO Сделка
  VALUES (3,1,-299,'01/08/2002')
```

Создаваемый *триггер* должен отреагировать на ее выполнение следующим образом: необходимо отменить команду, если в таблице Склад величина остатка товара оказалась меньше продаваемого количества товара с введенным кодом (в примере код товара=3). Во вставляемой записи количество товара указывается со знаком "+", если товар поставляется, и со знаком "-", если он продается. Представленный *триггер* настроен на обработку только одной добавляемой записи.

```
CREATE TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
  IF NOT EXISTS(SELECT *
     FROM inserted
     WHERE -inserted.количество<=ALL(SELECT
       Склад.Остаток
    FROM Склад,Сделка
    WHERE Склад.КодТовара=
```

```
      Сделка.КодТовара))
    BEGIN
      ROLLBACK TRAN
    PRINT
            'Отмена поставки: товара на складе нет'
    END
  END
```

Пример 14.1. Использование триггера для реализации ограничений на значение. (html, txt)

Пример 14.2. Использования *триггера* для сбора статистических данных.
Создать *триггер* для обработки операции вставки записи в таблицу Сделка, например, такой команды:

```
  INSERT INTO Сделка
    VALUES (3,1,200,'01/08/2002')
```

поставляется товар с кодом 3 от клиента с кодом 1 в количестве 200 единиц.
При продаже или получении товара необходимо соответствующим образом изменить количество его складского запаса. Если товара на складе еще нет, необходимо добавить соответствующую запись в таблицу Склад. *Триггер* обрабатывает только одну добавляемую строку.

```
  ALTER TRIGGER Триггер_ins
  ON Сделка FOR INSERT
  AS
  DECLARE @x INT, @y INT
  IF @@ROWCOUNT=1
  --в таблицу Сделка добавляется запись
  --о поставке товара
  BEGIN
  --количество проданного товара должно быть не
  --меньше, чем его остаток из таблицы Склад
  IF NOT EXISTS(SELECT *
           FROM inserted
           WHERE -inserted.количество<
    =ALL(SELECT Склад.Остаток
         FROM Склад,Сделка
         WHERE Склад.КодТовара=
             Сделка.КодТовара))
    BEGIN
      ROLLBACK TRAN
      PRINT 'откат товара нет '
    END
  --если записи о поставленном товаре еще нет,
  --добавляется соответствующая запись
  --в таблицу Склад
    IF NOT EXISTS ( SELECT *
           FROM Склад C, inserted i
           WHERE C.КодТовара=i.КодТовара )
      INSERT INTO Склад (КодТовара,Остаток)
  ELSE
  --если запись о товаре уже была в таблице
  --Склад, то определяется код и количество
```

```
--товара из добавленной в таблицу Сделка записи
 BEGIN
  SELECT @y=i.КодТовара, @x=i.Количество
  FROM Сделка C, inserted i
  WHERE C.КодТовара=i.КодТовара
--и производится изменения количества товара в
--таблице Склад
      UPDATE Склад
      SET Остаток=остаток+@x
      WHERE КодТовара=@y
  END
END
```

Пример 14.2. Использования триггера для сбора статистических данных. (html, txt)

Пример 14.3. Создать *триггер* для обработки операции удаления записи из таблицы Сделка, например, такой команды:

```
DELETE FROM Сделка WHERE КодСделки=4
```

Для товара, код которого указан при удалении записи, необходимо откорректировать его остаток на складе. *Триггер* обрабатывает только одну удаляемую запись.

```
CREATE TRIGGER Триггер_del
ON Сделка FOR DELETE
AS
IF @@ROWCOUNT=1 -- удалена одна запись
BEGIN
 DECLARE @y INT,@x INT
--определяется код и количество товара из
--удаленной из таблицы Склад записи
 SELECT @y=КодТовара, @x=Количество
 FROM deleted
--в таблице Склад корректируется количество
--товара
   UPDATE Склад
   SET Остаток=Остаток-@x
   WHERE КодТовара=@y
END
```

Пример 14.3. Триггер для обработки операции удаления записи из таблицы (html, txt)

Пример 14.4. Создать *триггер* для обработки операции изменения записи в таблице Сделка, например, такой командой:

```
UPDATE Сделка SET количество=количество-10
WHERE КодТовара=3
```

во всех сделках с товаром, имеющим код, равный 3, уменьшить количество товара на 10 единиц.

Указанная команда может привести к изменению сразу нескольких записей в таблице Сделка. Поэтому покажем, как создать *триггер*, обрабатывающий не одну запись. Для каждой измененной записи необходимо для старого (до изменения) кода товара уменьшить остаток товара на складе на величину старого (до изменения) количества товара и для нового (после изменения) кода товара увеличить его остаток на складе на величину нового (после изменения) значения. Чтобы обработать все измененные записи, введем курсоры, в которых сохраним все старые (из таблицы deleted) и все новые значения (из таблицы inserted).

```
CREATE TRIGGER Триггер_upd
ON Сделка FOR UPDATE
```

```
AS
DECLARE @x INT, @x_old INT, @y INT, @y_old INT
-- курсор с новыми значениями
DECLARE CUR1 CURSOR FOR
   SELECT КодТовара,Количество
   FROM inserted
-- курсор со старыми значениями
DECLARE CUR2 CURSOR FOR
   SELECT КодТовара,Количество
   FROM deleted
OPEN CUR1
OPEN CUR2
-- перемещаемся параллельно по обоим курсорам
   FETCH NEXT FROM CUR1 INTO @x, @y
   FETCH NEXT FROM CUR2 INTO @x_old, @y_old
   WHILE @@FETCH_STATUS=0
    BEGIN
--для старого кода товара уменьшается его
--количество на складе
     UPDATE Склад
     SET Остаток=Остаток-@y_old
     WHERE КодТовара=@x_old
--для нового кода товара, если такого товара
--еще нет на складе, вводится новая запись
     IF NOT EXISTS (SELECT * FROM Склад
       WHERE КодТовара=@x)
     INSERT INTO Склад(КодТовара,Остаток)
      VALUES (@x,@y)
     ELSE
--иначе для нового кода товара увеличивается
--его количество на складе
      UPDATE Склад
      SET Остаток=Остаток+@y
      WHERE КодТовара=@x
     FETCH NEXT FROM CUR1 INTO @x, @y
   FETCH NEXT FROM CUR2 INTO @x_old, @y_old
END
CLOSE CUR1
CLOSE CUR2
DEALLOCATE CUR1
DEALLOCATE CUR2
```

Пример 14.4. триггер для обработки операции изменения записи в таблице (html, txt)

В рассмотренном *триггере* отсутствует сравнение количества товара при изменении записи о сделке с его остатком на складе.

Пример 14.5. Исправим этот недостаток. Для генерирования сообщения об ошибке используем в теле *триггера* команду MS SQL Server RAISERROR, аргументами которой являются текст сообщения, уровень серьезности и статус ошибки.

```
ALTER TRIGGER Триггер_upd
ON Сделка FOR UPDATE
AS
```

```
DECLARE @x INT, @x_old INT, @y INT,
        @y_old INT ,@o INT
DECLARE CUR1 CURSOR FOR
   SELECT КодТовара,Количество
   FROM inserted
DECLARE CUR2 CURSOR FOR
   SELECT КодТовара,Количество
   FROM deleted
OPEN CUR1
OPEN CUR2
   FETCH NEXT FROM CUR1 INTO @x, @y
   FETCH NEXT FROM CUR2 INTO @x_old, @y_old
   WHILE @@FETCH_STATUS=0
    BEGIN
      SELECT @o=остаток
      FROM Склад
      WHERE кодтовара=@x
      IF @o<-@y
      BEGIN
        RAISERROR('откат',16,10)
        CLOSE CUR1
        CLOSE CUR2
        DEALLOCATE CUR1
        DEALLOCATE CUR22
        ROLLBACK TRAN
        RETURN
      END
        UPDATE Склад
      SET Остаток=Остаток-@y_old
      WHERE КодТовара=@x_old
      IF NOT EXISTS (SELECT * FROM Склад
        WHERE КодТовара=@x)
      INSERT INTO Склад(КодТовара,Остаток)
        VALUES (@x,@y)
    ELSE
      UPDATE Склад
      SET Остаток=Остаток+@y
      WHERE КодТовара=@x
   FETCH NEXT FROM CUR1 INTO @x, @y
   FETCH NEXT FROM CUR2 INTO @x_old, @y_old
END
CLOSE CUR1
CLOSE CUR2
DEALLOCATE CUR1
DEALLOCATE CUR2
```

Пример 14.5. Исправленный вариант триггера для обработки операции изменения записи в таблице (html, txt)

Пример 14.6. В примере 14.5 происходит отмена всех изменений при невозможности реализовать хотя бы одно из них. Создадим *триггер*, позволяющий отменять изменение только некоторых записей и выполнять изменение остальных.

В этом случае *триггер* выполняется не после изменения записей, а вместо команды изменения.

```
ALTER TRIGGER Триггер_upd
ON Сделка INSTEAD OF UPDATE
AS
DECLARE @k INT, @k_old INT
DECLARE @x INT, @x_old INT, @y INT
DECLARE @y_old INT ,@o INT
DECLARE CUR1 CURSOR FOR
   SELECT КодСделки, КодТовара,Количество
   FROM inserted
DECLARE CUR2 CURSOR FOR
   SELECT КодСделки, КодТовара,Количество
   FROM deleted
OPEN CUR1
OPEN CUR2
   FETCH NEXT FROM CUR1 INTO @k,@x, @y
   FETCH NEXT FROM CUR2 INTO @k_old,@x_old,
    @y_old
   WHILE @@FETCH_STATUS=0
    BEGIN
     SELECT @o=остаток
     FROM Склад
     WHERE КодТовара=@x
     IF @o>=-@y
     BEGIN
     RAISERROR('изменение',16,10)
     UPDATE Сделка SET количество=@y,
                    КодТовара=@x
     WHERE КодСделки=@k

     UPDATE Склад
     SET Остаток=Остаток-@y_old
     WHERE КодТовара=@x_old

     IF NOT EXISTS (SELECT * FROM Склад
       WHERE КодТовара=@x)
     INSERT INTO Склад(КодТовара, Остаток)
       VALUES (@x,@y)
     ELSE
     UPDATE Склад
     SET Остаток=Остаток+@y
     WHERE КодТовара=@x
    END
    ELSE
     RAISERROR('запись не изменена',16,10)
    FETCH NEXT FROM CUR1 INTO @k,@x, @y
    FETCH NEXT FROM CUR2 INTO @k_old,@x_old,
     @y_old
END
CLOSE CUR1
CLOSE CUR2
```

```
DEALLOCATE CUR1
DEALLOCATE CUR2
```

Лекция: Триггеры в рекурсивных структурах

1 Введение в рекурсивные структуры

Рассмотрим создание таблицы, реализующей *рекурсивную иерархию*, на примере данных, описывающих *отношения подчиненности* между сотрудниками. В таблице emp_mgr необходимо задать как имя сотрудника (emp), так и имя его начальника (mgr). Для *рекурсивной связи* одна и та же сущность является и родительской, и дочерней. При задании *рекурсивной связи* атрибут первичного ключа мигрирует в качестве внешнего ключа в состав неключевых атрибутов той же сущности (атрибуты emp – сотрудник и mgr – начальник таблицы emp_mgr). Информация о руководителе содержится в той же сущности, поскольку руководитель – сотрудник той же организации. Связь руководит/подчиняется (fk_emp) позволяет хранить древовидную иерархию подчиненности. Такой вид *рекурсивной связи* называется иерархической рекурсией и задает связь, когда руководитель (экземпляр родительской сущности) может иметь множество подчиненных (экземпляров дочерней сущности), но подчиненный – только одного руководителя. В среде MS SQL Server создадим таблицу emp_mgr:

```
CREATE TABLE emp_mgr
 (emp CHAR(2) PRIMARY KEY,
 mgr CHAR(2) NULL,
 NoOfReports INT DEFAULT 0,
 CONSTRAINT fk_emp FOREIGN KEY (mgr)
  REFERENCES emp_mgr (emp) )
```

В таблицу введено поле NoOfReports, в котором для каждого сотрудника определено количество его подчиненных.

Для удобства иллюстрации в качестве имени сотрудника и его начальника будут использоваться латинские буквы. Например, ввод данных в таблицу осуществляется операторами:

```
INSERT INTO emp_mgr(emp,mgr) VALUES('a',NULL)
INSERT INTO emp_mgr(emp,mgr) VALUES('b','a')
INSERT INTO emp_mgr(emp,mgr) VALUES('c','a')
INSERT INTO emp_mgr(emp,mgr) VALUES('d','a')
INSERT INTO emp_mgr(emp,mgr) VALUES('e','b')
INSERT INTO emp_mgr(emp,mgr) VALUES('f','b')
INSERT INTO emp_mgr(emp,mgr) VALUES('g','b')
INSERT INTO emp_mgr(emp,mgr) VALUES('i','c')
INSERT INTO emp_mgr(emp,mgr) VALUES('k','d')
```

После ввода данных в таблицу emp_mgr оператор SELECT * FROM emp_mgr возвращает следующий результат:

```
emp   mgr   NoOfReports
-------------------------
a     NULL  3
b     a     3
c     a     1
d     a     1
e     b     0
```

```
f    b     0
g    b     0
i    c     0
k    d     0
```

2 Реализация правил целостности данных

Целостность, непротиворечивость и достоверность информации в таблицах с *рекурсивными связями* обеспечиваются выполнением ряда правил:

1. Каждый сотрудник имеет только одного руководителя.
2. Каждый сотрудник не является сам себе руководителем.
3. Каждый руководитель в первую очередь сотрудник.
4. Имеется только один сотрудник (директор организации), который никому не подчиняется.
5. Правило 2 необходимо усилить. Каждый сотрудник не должен находиться в роли собственного руководителя не только непосредственно, но и опосредствованно, через других сотрудников.

Выполнение правила 1 обеспечивается ограничением первичного ключа и не требует дополнительных SQL-операторов.
Рассмотрим правило 2. Имена сотрудника и его начальника в одной записи не должны совпадать. При добавлении и изменении записи в таблице emp_mgr это требование предъявляется к новой записи, которая до подтверждения транзакции располагается во временной таблице с именем inserted. Этому правилу соответствуют следующие SQL-операторы:

```
IF EXISTS (SELECT * FROM inserted
        WHERE mgr=emp)
BEGIN
  ROLLBACK TRAN
  RAISERROR('САМ СЕБЕ НАЧАЛЬНИК',16,10)
  RETURN
END
```

Правило 3 говорит о том, что именем начальника может быть только уже внесенное в таблицу имя сотрудника. Это требование представляет собой декларативную ссылочную целостность и обеспечивается ограничением внешнего ключа. Однако, чтобы запустить механизм триггеров, придется удалить ограничение внешнего ключа и его функцию возложить на триггер.
В новой или измененной записи имя начальника должно быть указано и уже присутствовать в таблице в качестве имени сотрудника, что может быть записано следующими SQL-операторами:

```
IF EXISTS(SELECT * FROM inserted
   WHERE mgr IS NOT NULL) AND
   NOT EXISTS(SELECT * FROM inserted,emp_mgr
   WHERE emp_mgr.emp=inserted.mgr)
BEGIN
  RAISERROR('НЕТ НАЧАЛЬНИКА',16,10)
  ROLLBACK TRAN
  RETURN
END
```

или (что эквивалентно)

```
IF NOT EXISTS(SELECT * FROM emp_mgr, inserted
```

```
    WHERE emp_mgr.emp=inserted.mgr
    OR inserted.mgr IS NULL)
  BEGIN
   RAISERROR('НЕТ НАЧАЛЬНИКА',16,10)
   ROLLBACK TRAN
   RETURN
  END
```

В соответствии с правилом 4 необходимо проверить, введена ли запись о директоре (сотруднике, у которого нет начальника). Если такая запись уже есть, ввод нового директора запрещается с помощью следующих SQL-операторов:

```
  IF EXISTS (SELECT * FROM inserted
          WHERE mgr IS NULL)
     AND EXISTS
            (SELECT * FROM emp_mgr,inserted
     WHERE emp_mgr.mgr IS NULL
     AND emp_mgr.emp<>inserted.emp)
  BEGIN
   ROLLBACK TRAN
   RAISERROR('ОДИН ДИРЕКТОР УЖЕ ЕСТЬ',16,10)
   RETURN
  END
```

Оператор UPDATE может изменить иерархическую структуру таким образом, что возникает ситуация, когда сотрудник становится начальником самому себе через других сотрудников, т.е. в иерархии подчиненности возникает петля. Для исключения подобных преобразований используем SQL-операторы:

```
  IF UPDATE(mgr)--изменился начальник
  BEGIN
   DECLARE @x CHAR(2), @y CHAR(2), @xx CHAR(2)
  --узнали имя сотрудника,
  --у которого изменился начальник
   SELECT @xx=inserted.emp FROM inserted
   SELECT @x=@xx
   SELECT @y='*'
   WHILE @y IS NOT NULL
  --пока не дошли до директора
   BEGIN
  --запомнили имя начальника
   SELECT @y=mgr FROM emp_mgr
   WHERE emp=@x
   IF @xx=@y
  --имя сотрудника и его начальника совпали
   BEGIN
   RAISERROR('транзитивное замыкание',16,10)
    ROLLBACK TRAN
    RETURN
   END
   ELSE
  --далее начальник становится сотрудником,
  --и в цикле будем искать его начальника
    SELECT @x=@y
   END
```

```
  END
```
Чтобы сработали триггеры, необходимо удалить ограничение внешнего ключа:
```
  ALTER TABLE emp_mgr DROP CONSTRAINT fk_emp
```
Ниже приведен текст триггеров, поддерживающих целостность данных в иерархических структурах. Предполагается, что триггеры обрабатывают ввод, изменение или удаление одной записи.

3 Добавление записи в рекурсивную структуру

```
  ALTER TRIGGER emp_ins
  ON emp_mgr FOR INSERT
  AS
  --Правило 2
  IF EXISTS (SELECT * FROM inserted WHERE mgr=emp)
  BEGIN
     ROLLBACK TRAN
     RAISERROR('САМ СЕБЕ НАЧАЛЬНИК',16,10)
     RETURN
  END
  --Правило 4
  IF EXISTS (SELECT * FROM inserted WHERE mgr IS NULL) AND
     EXISTS (SELECT * FROM emp_mgr,inserted
        WHERE emp_mgr.mgr IS NULL
                    AND emp_mgr.emp<>inserted.emp)
  BEGIN
     ROLLBACK TRAN
     RAISERROR('ОДИН ДИРЕКТОР УЖЕ ЕСТЬ',16,10)
     RETURN
  END
  --Правило 3
  IF EXISTS(SELECT * FROM inserted
     WHERE mgr IS NOT NULL) AND
  NOT EXISTS(SELECT * FROM inserted,emp_mgr
     WHERE emp_mgr.emp=inserted.mgr)
  BEGIN
    RAISERROR('НЕТ ТАКОГО НАЧАЛЬНИКА',16,10)
    ROLLBACK TRAN
    RETURN
  END
  --Пересчет числа подчиненных у начальника
  --добавленного подчиненного
  DECLARE @e CHAR(2), @m CHAR(2)
    SELECT @e=emp_mgr.emp FROM emp_mgr, inserted
    WHERE emp_mgr.emp=inserted.mgr
    UPDATE emp_mgr
    SET emp_mgr.NoOfReports=emp_mgr.NoOfReports+1
    WHERE emp_mgr.emp=@e
```
Пример 15.1. Триггер для добавления записи в таблицу. (html, txt)

Изменение записи в рекурсивной структуре

```
CREATE TRIGGER emp_upd ON emp_mgr
FOR UPDATE
AS
IF UPDATE(mgr)
BEGIN
--Правило 5
DECLARE @x CHAR(2), @y CHAR(2), @xx CHAR(2)
SELECT @xx=inserted.emp FROM inserted
SELECT @x=@xx
SELECT @y='*'
WHILE @y IS NOT NULL
 BEGIN
 SELECT @y=mgr FROM emp_mgr WHERE emp=@x
 IF @xx=@y
  BEGIN
   RAISERROR('транзитивное замыкание',16,10)
   ROLLBACK TRAN
   RETURN
   END
  ELSE
  SELECT @x=@y
 END
END
--Правило 2
IF EXISTS (SELECT * FROM inserted WHERE mgr=emp)
BEGIN
 ROLLBACK TRAN
 RAISERROR('САМ СЕБЕ НАЧАЛЬНИК',16,10)
 RETURN
END
--Правило 4
IF EXISTS (SELECT * FROM inserted WHERE mgr
 IS NULL) AND EXISTS (SELECT *
 FROM emp_mgr,inserted WHERE emp_mgr.mgr IS NULL
 AND emp_mgr.emp<>inserted.emp)
BEGIN
 ROLLBACK TRAN
 RAISERROR('ОДИН ДИРЕКТОР УЖЕ ЕСТЬ',16,10)
 RETURN
END
--Правило 3
IF UPDATE(mgr)
 IF NOT EXISTS(SELECT * FROM emp_mgr, inserted
  WHERE emp_mgr.emp=inserted.mgr
  OR inserted.mgr IS NULL)
BEGIN
 RAISERROR('НЕТ ТАКОГО НАЧАЛЬНИКА',16,10)
 ROLLBACK TRAN
 RETURN
END
IF UPDATE(mgr)
```

```
--пересчет числа подчиненных у старого и нового
--начальников
BEGIN
  UPDATE emp_mgr
    SET emp_mgr.NoOfReports=emp_mgr.NoOfReports+1
    FROM inserted WHERE emp_mgr.emp=inserted.mgr
    UPDATE emp_mgr
    SET emp_mgr.NoOfReports=emp_mgr.NoOfReports-1
    FROM deleted WHERE emp_mgr.emp=deleted.mgr
END
IF UPDATE(emp)
--если изменилось имя сотрудника, следует изменить
--имя начальника у всех его подчиненных
UPDATE emp_mgr SET emp_mgr.mgr=inserted.emp
FROM emp_mgr, inserted, deleted WHERE
emp_mgr.mgr=deleted.emp
```

Пример 15.2. Триггер для изменения записи в таблице. (html, txt)

Попытка подчинить сотрудника с именем ‘b’ начальнику с именем ‘e’ будет сервером отвергнута, иначе в организации сложилась бы такая ситуация: сотрудник ‘e’ подчинятся сотруднику ‘b’, а сотрудник ‘b’ подчиняется сотруднику ‘e’.

```
UPDATE emp_mgr SET mgr='e' WHERE emp='b'
Server: Msg 50000, Level 16, State 10,
  Procedure emp_upd,
  Line 15 транзитивное замыкание
```

Выполнение команды

```
UPDATE emp_mgr SET mgr='f' WHERE emp='e'
```

и команды

```
UPDATE emp_mgr SET mgr='a' WHERE emp='g'
```

приведет к следующему изменению первоначальной иерархической структуры:

```
emp   mgr   NoOfReports
------------------------
a     NULL  4
b     a     1
c     a     1
d     a     1
e     f     0
f     b     1
g     a     0
i     c     0
k     d     0
```

Удаление записи из рекурсивной структуры

```
ALTER TRIGGER emp_del
ON emp_mgr
FOR DELETE
AS
DECLARE @e CHAR(2), @m CHAR(2), @r INT
SELECT @e=emp,@m=mgr,@r=NoOfReports FROM deleted
IF @m IS NOT NULL
```

```
-- удаляется сотрудник, не являющийся директором
 BEGIN
   IF @r=0
-- удаляется сотрудник, у которого нет подчиненных
-- уменьшается число подчиненных у начальника
-- удаляемого сотрудника
   UPDATE emp_mgr SET NoOfReports=
                NoOfReports-1
   WHERE emp=@m
  ELSE
    BEGIN
-- удаляется сотрудник, у которого есть подчиненные
-- переподчиним его подчиненных его начальнику,
-- т.е. начальником подчиненных удаляемого сотрудника
-- становится его начальник
   UPDATE emp_mgr SET NoOfReports=
                NoOfReports+@r-1
   WHERE emp=@m
      UPDATE emp_mgr SET mgr=@m
      WHERE mgr=@e
    END
 END
 ELSE
-- Правило 4
   IF EXISTS(SELECT * FROM emp_mgr)
     BEGIN
       ROLLBACK TRAN
            RAISERROR('НЕЛЬЗЯ УДАЛЯТЬ
                  ДИРЕКТОРА',16,10)
       RETURN
     END
```

Пример 15.3. Триггер для удаления записи из таблицы. (html, txt)

Попытка удаления записи о директоре будет отвергнута сервером:

```
DELETE FROM emp_mgr WHERE emp='a'
Server: Msg 50000, Level 16, State 10,
  Procedure emp_del, Line 24
НЕЛЬЗЯ УДАЛЯТЬ ДИРЕКТОРА
```

В результате удаления рядового сотрудника с именем b его подчиненные e, f и g станут подчиненными сотрудника с именем a.

```
DELETE FROM emp_mgr WHERE emp='b'
```

Первоначальное содержимое таблицы emp_mgr изменится следующим образом:

```
emp    mgr    NoOfReports
-------------------------
a      NULL   5
c      a      1
d      a      1
e      a      0
f      a      0
g      a      0
i      c      0
```

k d 0

Лекция: Транзакции и блокировки

1 Введение в транзакции

Концепция *транзакций* – неотъемлемая часть любой клиент-серверной базы данных.
Под ***транзакцией*** понимается *неделимая* с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), приводящая к одному из двух возможных результатов: либо последовательность выполняется, если все операторы правильные, либо вся *транзакция* откатывается, если хотя бы один оператор не может быть успешно выполнен. Обработка *транзакций* гарантирует целостность информации в базе данных. Таким образом, *транзакция* переводит базу данных из одного целостного состояния в другое.
Поддержание механизма *транзакций* – показатель уровня развитости СУБД. Корректное поддержание *транзакций* одновременно является основой обеспечения целостности БД.
Транзакции также составляют основу *изолированности* в многопользовательских системах, где с одной БД параллельно могут работать несколько пользователей или прикладных программ.
Одна из основных задач СУБД – обеспечение *изолированности*, т.е. создание такого режима функционирования, при котором каждому пользователю казалось бы, что БД доступна только ему. Такую задачу СУБД принято называть параллелизмом *транзакций*.
Большинство выполняемых действий производится в теле *транзакций*. По умолчанию каждая команда выполняется как самостоятельная *транзакция*. При необходимости пользователь может явно указать ее *начало* и *конец*, чтобы иметь возможность включить в нее несколько команд.
При выполнении *транзакции* система управления базами данных должна придерживаться определенных правил обработки набора команд, входящих в *транзакцию*. В частности, разработано четыре правила, известные как требования ACID, они гарантируют правильность и надежность работы системы.

2 ACID-свойства транзакций

Характеристики *транзакций* описываются в терминах ACID (Atomicity, Consistency, Isolation, Durability – *неделимость*, *согласованность*, *изолированность*, *устойчивость*).

- *Транзакция* ***неделима*** в том смысле, что представляет собой единое целое. Все ее компоненты либо имеют место, либо нет. Не бывает частичной *транзакции*. Если может быть выполнена лишь часть *транзакции*, она отклоняется.
- *Транзакция* является ***согласованной***, потому что не нарушает бизнес-логику и отношения между элементами данных. Это *свойство* очень важно при разработке клиент-серверных систем, поскольку в хранилище данных поступает большое количество *транзакций* от разных систем и объектов. Если хотя бы одна из них нарушит целостность данных, то все остальные могут выдать неверные результаты.
- *Транзакция* всегда ***изолирована***, поскольку ее результаты самодостаточны. Они не зависят от предыдущих или последующих *транзакций* – это *свойство* называется *сериализуемостью* и означает, что *транзакции* в последовательности независимы.
- *Транзакция* ***устойчива***. После своего завершения она сохраняется в системе, которую ничто не может вернуть в исходное (до *начала транзакции*) состояние, т.е. происходит фиксация *транзакции*, означающая, что ее действие постоянно даже при сбое системы.

При этом подразумевается некая форма хранения информации в постоянной памяти как часть *транзакции*.

Указанные выше правила выполняет сервер. Программист лишь выбирает нужный *уровень изоляции*, заботится о соблюдении логической целостности данных и бизнес-правил. На него возлагаются обязанности по созданию эффективных и логически верных алгоритмов обработки данных. Он решает, какие команды должны выполняться как одна *транзакция*, а какие могут быть разбиты на несколько последовательно выполняемых *транзакций*. Следует по возможности использовать небольшие *транзакции*, т.е. включающие как можно меньше команд и изменяющие минимум данных. Соблюдение этого требования позволит наиболее эффективным образом обеспечить одновременную работу с данными множества пользователей.

3 Блокировки

Повышение эффективности работы при использовании небольших *транзакций* связано с тем, что при выполнении *транзакции* сервер накладывает на данные *блокировки*.
Блокировкой называется временное ограничение на выполнение некоторых операций обработки данных. *Блокировка* может быть наложена как на отдельную строку таблицы, так и на всю базу данных. ***Управлением блокировками*** на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. *Транзакции* и *блокировки* тесно связаны друг с другом. *Транзакции* накладывают *блокировки* на данные, чтобы обеспечить выполнение требований ACID. Без использования блокировок несколько *транзакций* могли бы изменять одни и те же данные.
Блокировка представляет собой метод *управления параллельными процессами*, при котором объект БД не может быть модифицирован без ведома *транзакции*, т.е. происходит блокирование доступа к объекту со стороны других *транзакций*, чем исключается непредсказуемое изменение объекта. Различают два вида *блокировки*:

- *блокировка* записи – *транзакция* блокирует строки в таблицах таким образом, что запрос другой *транзакции* к этим строкам будет *отменен*;
- *блокировка* чтения – *транзакция* блокирует строки так, что запрос со стороны другой *транзакции* на *блокировку* записи этих строк будет отвергнут, а на *блокировку* чтения – принят.

В СУБД используют протокол доступа к данным, позволяющий избежать проблемы параллелизма. Его суть заключается в следующем:

- *транзакция*, результатом действия которой на строку данных в таблице является ее извлечение, обязана наложить *блокировку* чтения на эту строку;
- *транзакция*, предназначенная для модификации строки данных, накладывает на нее *блокировку* записи;
- если запрашиваемая *блокировка* на строку отвергается из-за уже имеющейся *блокировки*, то *транзакция* переводится в режим ожидания до тех пор, пока *блокировка* не будет снята;
- *блокировка* записи сохраняется вплоть до конца выполнения *транзакции*.

Решение ***проблемы параллельной обработки*** БД заключается в том, что строки таблиц блокируются, а последующие *транзакции*, модифицирующие эти строки, отвергаются и переводятся в режим ожидания. В связи со *свойством* сохранения целостности БД *транзакции* являются подходящими единицами *изолированности* пользователей. Действительно, если каждый сеанс взаимодействия с базой данных реализуется *транзакцией*, то пользователь

начинает с того, что обращается к *согласованному* состоянию базы данных – состоянию, в котором она могла бы находиться, даже если бы пользователь работал с ней в одиночку. Если в системе управления базами данных не реализованы механизмы блокирования, то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могут возникнуть следующие проблемы одновременного доступа:

- проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т.к. каждая последующая *транзакция* перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;
- проблема ***"грязного" чтения*** возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;
- проблема ***неповторяемого чтения*** является следствием неоднократного считывания *транзакцией* одних и тех же данных. Во время выполнения первой *транзакции* другая может внести в данные изменения, поэтому при повторном чтении первая *транзакция* получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;
- проблема чтения ***фантомов*** появляется после того, как одна *транзакция* выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в специально разработанном стандарте определены четыре *уровня блокирования*. *Уровень изоляции транзакции* определяет, могут ли другие (конкурирующие) *транзакции* вносить изменения в данные, измененные текущей *транзакцией*, а также может ли текущая *транзакция* видеть изменения, произведенные конкурирующими *транзакциями*, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

- *уровень* 0 – запрещение "загрязнения" данных. Этот *уровень* требует, чтобы изменять данные могла только одна *транзакция*; если другой *транзакции* необходимо изменить те же данные, она должна ожидать завершения первой *транзакции*;
- *уровень* 1 – запрещение *"грязного" чтения*. Если *транзакция* начала изменение данных, то никакая другая *транзакция* не сможет прочитать их до завершения первой;
- *уровень* 2 – запрещение *неповторяемого чтения*. Если *транзакция* считывает данные, то никакая другая *транзакция* не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;
- *уровень* 3 – запрещение *фантомов*. Если *транзакция* обращается к данным, то никакая другая *транзакция* не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении *транзакции*. Реализация этого *уровня блокирования* выполняется путем использования блокировок диапазона ключей. Подобная *блокировка* накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.
-

4 Управление транзакциями

Под ***управлением транзакциями*** понимается способность управлять различными операциями над данными, которые выполняются внутри реляционной СУБД. Прежде всего, имеется в виду выполнение операторов INSERT, UPDATE и DELETE. Например, после создания таблицы (выполнения оператора CREATE TABLE) не нужно фиксировать результат: создание таблицы фиксируется в базе данных автоматически. Точно так же с помощью *отмены транзакции* не удастся восстановить только что удаленную оператором DROP TABLE таблицу.
После успешного выполнения команд, заключенных в тело одной *транзакции*, немедленного изменения данных не происходит. Для окончательного завершения *транзакции* существуют так называемые команды *управления транзакциями*, с помощью которых можно либо сохранить в базе данных все изменения, произошедшие в ходе ее выполнения, либо полностью их *отменить*.
Существуют три команды, которые используются для *управления транзакциями*:

- COMMIT – для *сохранения изменений*;
- ROLLBACK – для *отмены изменений*;
- SAVEPOINT – для *установки особых точек возврата*.

После завершения *транзакции* вся информация о произведенных изменениях хранится либо в специально выделенной оперативной памяти, либо во временной области отката в самой базе данных до тех пор, пока не будет выполнена одна из команд *управления транзакциями*. Затем все изменения или фиксируются в базе данных, или отбрасываются, а временная область отката освобождается.
Команда COMMIT предназначена для ***сохранения*** в базе данных всех изменений, произошедших в ходе выполнения *транзакции*. Она сохраняет результаты всех операций, которые имели место после выполнения последней команды COMMIT или ROLLBACK.
Команда ROLLBACK предназначена для ***отмены транзакций***, еще не сохраненных в базе данных. Она *отменяет* только те *транзакции*, которые были выполнены с момента выдачи последней команды COMMIT или ROLLBACK.
Команда SAVEPOINT (точка сохранения) предназначена для установки в *транзакции* особых точек, куда в дальнейшем может быть произведен откат (при этом отката всей *транзакции* не происходит). Команда имеет следующий вид:

```
SAVEPOINT имя_точки_сохранения
```

Она служит исключительно для создания точек сохранения среди операторов, предназначенных для изменения данных. Имя точки сохранения в связанной с ней группе *транзакций* должно быть уникальным.
Для *отмены* действия группы *транзакций*, ограниченных точками сохранения, используется команда ROLLBACK со следующим синтаксисом:

```
ROLLBACK TO имя_точки_сохранения
```

Поскольку с помощью команды SAVEPOINT крупное число *транзакций* может быть разбито на меньшие и поэтому более управляемые группы, ее применение является одним из способов *управления транзакциями*.

5 Управление транзакциями в среде MS SQL Server

Определение транзакций

SQL Server предлагает множество средств *управления поведением транзакций*. Пользователи в основном должны указывать только *начало* и *конец транзакции*, используя команды SQL или API (прикладного интерфейса программирования). *Транзакция* определяется на уровне соединения с базой данных и при закрытии соединения автоматически закрывается. Если пользователь попытается установить соединение снова и продолжить выполнение *транзакции*,

то это ему не удастся. Когда *транзакция* начинается, все команды, выполненные в соединении, считаются телом одной *транзакции*, пока не будет достигнут ее *конец*.
SQL Server поддерживает три вида определения *транзакций*:

- *явное*;
- автоматическое;
- подразумеваемое.

По умолчанию SQL Server работает в режиме автоматического *начала транзакций*, когда каждая команда рассматривается как отдельная *транзакция*. Если команда выполнена успешно, то ее изменения фиксируются. Если при выполнении команды произошла ошибка, то сделанные изменения *отменяются* и система возвращается в первоначальное состояние.
Когда пользователю понадобится создать *транзакцию*, включающую несколько команд, он должен явно указать *транзакцию*.
Сервер работает только в одном из двух режимов определения *транзакций*: автоматическом или подразумевающемся. Он не может находиться в режиме исключительно явного определения *транзакций*. Этот режим работает поверх двух других.
Для установки режима автоматического определения *транзакций* используется команда:

```
SET IMPLICIT_TRANSACTIONS  OFF
```

При работе в режиме *неявного* (подразумевающегося) *начала транзакций* SQL Server автоматически начинает новую *транзакцию*, как только завершена предыдущая. Установка режима подразумевающегося определения *транзакций* выполняется посредством другой команды:

```
SET IMPLICIT_TRANSACTIONS  ON
```

Явные транзакции

Явные транзакции требуют, чтобы пользователь указал *начало* и *конец транзакции*, используя следующие команды:

- *начало транзакции*: в журнале *транзакций* фиксируются первоначальные значения изменяемых данных и момент *начала транзакции*;
  - BEGIN TRAN[SACTION]
  - [имя_транзакции |
  - @имя_переменной_транзакции
  - [WITH MARK [‘описание_транзакции’]]]
- *конец транзакции*: если в теле *транзакции* не было ошибок, то эта команда предписывает серверу зафиксировать все изменения, сделанные в *транзакции*, после чего в журнале *транзакций* помечается, что изменения зафиксированы и *транзакция* завершена;
  - COMMIT [TRAN[SACTION]
  - [имя_транзакции |
  - @имя_переменной_транзакции]]
- создание внутри *транзакции* точки сохранения: СУБД сохраняет состояние БД в текущей точке и присваивает сохраненному состоянию имя точки сохранения;
  - SAVE TRAN[SACTION]
  - {имя_точки_сохранеия |
  - @имя_переменной_точки_сохранения}
- прерывание *транзакции*; когда сервер встречает эту команду, происходит откат *транзакции*, восстанавливается первоначальное состояние системы и в журнале *транзакций* отмечается, что *транзакция* была *отменена*. Приведенная ниже команда *отменяет все изменения*, сделанные в БД после оператора BEGIN TRANSACTION или

отменяет изменения, сделанные в БД после точки сохранения, возвращая *транзакцию* к месту, где был выполнен оператор SAVE TRANSACTION.

- ROLLBACK [TRAN[SACTION]
- [имя_транзакции |
- @имя_переменной_транзакции
- | имя_точки_сохранения
- |@имя_переменной_точки_сохранения]]

Функция @@TRANCOUNT возвращает количество активных транзакций.
Функция @@NESTLEVEL возвращает уровень *вложенности транзакций*.

```
BEGIN TRAN
SAVE TRANSACTION  point1
```

Пример 16.1. Использование точек сохранения (html, txt)

В точке point1 сохраняется первоначальное состояние таблицы Товар

```
DELETE FROM Товар WHERE КодТовара=2
        SAVE TRANSACTION  point2
```

В точке point2 сохраняется состояние таблицы Товар без товаров с кодом 2.

```
DELETE FROM Товар WHERE КодТовара=3
        SAVE TRANSACTION  point3
```

В точке point3 сохраняется состояние таблицы Товар без товаров с кодом 2 и с кодом 3.

```
DELETE FROM Товар WHERE КодТовара<>1
        ROLLBACK TRANSACTION point3
```

Происходит возврат в состояние таблицы без товаров с кодами 2 и 3, отменяется последнее удаление.

```
SELECT * FROM Товар
```

Оператор SELECT покажет таблицу Товар без товаров с кодами 2 и 3.

```
ROLLBACK TRANSACTION point1
```

Происходит возврат в первоначальное состояние таблицы.

```
SELECT * FROM Товар
COMMIT
```

Первоначальное состояние сохраняется.

6 Вложенные транзакции

Вложенными называются *транзакции*, выполнение которых инициируется из тела уже активной *транзакции*.
Для создания *вложенной транзакции* пользователю не нужны какие-либо дополнительные команды. Он просто начинает новую *транзакцию*, не закрыв предыдущую. Завершение *транзакции* верхнего уровня откладывается до завершения *вложенных транзакций*. Если *транзакция* самого нижнего (*вложенного*) уровня завершена неудачно и *отменена*, то все *транзакции* верхнего уровня, включая *транзакцию* первого уровня, будут *отменены*. Кроме того, если несколько *транзакций* нижнего уровня были завершены успешно (но не зафиксированы), однако на среднем уровне (не самая верхняя *транзакция*) неудачно завершилась другая *транзакция*, то в соответствии с требованиями ACID произойдет откат всех *транзакций* всех уровней, включая успешно завершенные. Только когда все *транзакции* на всех уровнях завершены успешно, происходит фиксация всех сделанных изменений в результате успешного завершения *транзакции* верхнего уровня.
Каждая команда COMMIT TRANSACTION работает только с последней начатой *транзакцией*. При завершении *вложенной транзакции* команда COMMIT применяется к наиболее "глубокой" *вложенной транзакции*. Даже если в команде COMMIT TRANSACTION указано имя *транзакции* более высокого уровня, будет завершена *транзакция*, начатая последней.

Если команда ROLLBACK TRANSACTION используется на любом уровне *вложенности* без указания имени *транзакции*, то откатываются все *вложенные транзакции*, включая *транзакцию* самого высокого (верхнего) уровня. В команде ROLLBACK TRANSACTION разрешается указывать только имя самой верхней *транзакции*. Имена любых *вложенных транзакций* игнорируются, и попытка их указания приведет к ошибке. Таким образом, при откате *транзакции* любого уровня *вложенности* всегда происходит откат всех *транзакций*. Если же требуется откатить лишь часть *транзакций*, можно использовать команду SAVE TRANSACTION , с помощью которой создается точка сохранения.

```
BEGIN TRAN
INSERT Товар (Название, остаток)
VALUES ('v',40)
  BEGIN TRAN
  INSERT Товар (Название, остаток)
  VALUES ('n',50)
  BEGIN TRAN
        INSERT Товар (Название, остаток)
        VALUES ('m',60)
ROLLBACK TRAN
```

Пример 16.2. Вложенные транзакции. (html, txt)

Здесь происходит возврат на начальное состояние таблицы, поскольку выполнение команды ROLLBACK TRAN без указания имени *транзакции* откатывает все *транзакции*.

7 Блокировки в среде MS SQL Server

Управление блокировками

Пользователю чаще всего не нужно предпринимать никаких действий по *управлению блокировками*. Всю работу по установке, снятию и разрешению конфликтов выполняет специальный компонент сервера, называемый менеджером блокировок. MS SQL Server поддерживает различные ***уровни блокирования*** объектов (или детализацию блокировок), начиная с отдельной строки таблицы и заканчивая базой данных в целом. Менеджер блокировок автоматически оценивает, какое количество данных необходимо блокировать, и устанавливает соответствующий тип *блокировки*. Это позволяет поддерживать равновесие между производительностью работы системы блокирования и возможностью пользователей получать доступ к данным. ***Блокирование на уровне строки*** позволяет наиболее точно управлять таким доступом, поскольку блокируются только действительно изменяемые строки. Множество пользователей могут одновременно работать с данными с минимальными задержками. Платой за это является увеличение числа операций установки и снятия блокировок, а также большое количество служебной информации, которое приходится хранить для отслеживания установленных блокировок. При ***блокировке на уровне таблицы*** производительность системы блокирования резко увеличивается, так как необходимо установить лишь одну *блокировку* и снять ее только после завершения *транзакции*. Пользователь при этом имеет максимальную скорость доступа к данным. В то же время они не доступны никому другому, потому что вся таблица заблокирована. Приходится ожидать, пока текущий пользователь завершит работу.

Действия, выполняемые пользователями при работе с данными, сводятся к операциям двух типов: их чтению и изменению. В операции по изменению включаются действия по добавлению, удалению и собственно изменению данных. В зависимости от выполняемых действий сервер накладывает определенный тип *блокировки* из следующего перечня:

- ***Коллективные блокировки***. Они накладываются при выполнении операций чтения данных (например, SELECT). Если сервер установил на ресурс *коллективную блокировку*, то пользователь может быть уверен, что уже никто не сможет изменить эти данные.
- ***Блокировка обновления***. Если на ресурс установлена *коллективная блокировка* и для этого ресурса устанавливается *блокировка обновления*, то никакая *транзакция* не сможет наложить *коллективную блокировку* или *блокировку обновления*.
- ***Монопольная блокировка***. Этот тип блокировок используется, если *транзакция* изменяет данные. Когда сервер устанавливает *монопольную блокировку* на ресурс, то никакая другая *транзакция* не может прочитать или изменить заблокированные данные. *Монопольная блокировка* не совместима ни с какими другими *блокировками*, и ни одна *блокировка*, включая *монопольную*, не может быть наложена на ресурс.
- *Блокировка* массивного обновления. Накладывается сервером при выполнении операций массивного копирования в таблицу и запрещает обращение к таблице любым другим процессам. В то же время несколько процессов, выполняющих массивное копирование, могут одновременно вставлять строки в таблицу.

Помимо перечисленных основных типов блокировок SQL Server поддерживает ряд ***специальных блокировок***, предназначенных для повышения производительности и функциональности обработки данных. Они называются *блокировками* намерений и используются сервером в том случае, если *транзакция* намеревается получить доступ к данным вниз по иерархии и для других *транзакций* необходимо установить запрет на наложение блокировок, которые будут конфликтовать с *блокировкой*, накладываемой первой *транзакцией*. Ранее рассмотренные *блокировки* относятся к данным. Помимо перечисленных в среде SQL Server существует два других типа блокировок: *блокировка* диапазона ключей и *блокировка* схемы (метаданных, описывающих структуру объекта).
Блокировка диапазона ключей решает проблему возникновения *фантомов* и обеспечивает требования *сериализуемости транзакции*. *Блокировки* этого типа устанавливаются на диапазон строк, соответствующих определенному логическому условию, с помощью которого осуществляется выборка данных из таблицы.
Блокировка схемы используется при выполнении команд модификации структуры таблиц для обеспечения целостности данных.

"Мертвые" блокировки

"Мертвые", или тупиковые, *блокировки* характерны для многопользовательских систем. *"Мертвая" блокировка* возникает, когда две *транзакции* блокируют два блока данных и для завершения любой из них нужен доступ к данным, заблокированным ранее другой *транзакцией*. Для завершения каждой *транзакции* необходимо дождаться, пока блокированная другой *транзакцией* часть данных будет разблокирована. Но это невозможно, так как вторая *транзакция* ожидает разблокирования ресурсов, используемых первой.
Без применения специальных механизмов обнаружения и снятия *"мертвых" блокировок* нормальная работа *транзакций* будет нарушена. Если в системе установлен бесконечный период ожидания завершения *транзакции* (а это задано по умолчанию), то при возникновении *"мертвой" блокировки* для двух *транзакций* вполне возможно, что, ожидая освобождения заблокированных ресурсов, в тупике окажутся и новые *транзакции*. Чтобы избежать подобных проблем, в среде MS SQL Server реализован специальный механизм разрешения конфликтов тупикового блокирования.
Для этих целей сервер снимает одну из блокировок, вызвавших конфликт, и откатывает инициализировавшую ее *транзакцию*. При выборе *блокировки*, которой необходимо пожертвовать, сервер исходит из соображений минимальной стоимости.

Полностью избежать возникновения *"мертвых" блокировок* нельзя. Хотя сервер и имеет эффективные механизмы снятия таких блокировок, все же при написании приложений следует учитывать вероятность их возникновения и предпринимать все возможные действия для предупреждения этого. *"Мертвые" блокировки* могут существенно снизить производительность, поскольку системе требуется достаточно много времени для их обнаружения, отката *транзакции* и повторного ее выполнения.
Для минимизации возможности образования *"мертвых" блокировок* при разработке кода *транзакции* следует придерживаться следующих правил:

- выполнять действия по обработке данных в постоянном порядке, чтобы не создавать условия для захвата одних и тех же данных;
- избегать взаимодействия с пользователем в теле *транзакции*;
- минимизировать длительность *транзакции* и выполнять ее по возможности в одном пакете;
- применять как можно более низкий *уровень изоляции*.

8 Уровни изоляции SQL Server

Уровень изоляции определяет степень независимости *транзакций* друг от друга. Наивысшим *уровнем изоляции* является ***сериализуемость***, обеспечивающая полную независимость *транзакций* друг от друга. Каждый последующий уровень соответствует требованиям всех предыдущих и обеспечивает дополнительную защиту *транзакций*.
SQL Server поддерживает все четыре *уровня изоляции*, определенные стандартом ANSI. *Уровень изоляции* устанавливается командой:

```
SET TRANSACTION ISOLATION LEVEL
        { READ COMMITTED
        | READ UNCOMMITTED
        | REPEATABLE READ
        | SERIALIZABLE }
```

- READ UNCOMMITED – ***незавершенное чтение***, или допустимо черновое чтение. Низший *уровень изоляции*, соответствующий уровню 0. Он гарантирует только физическую целостность данных: если несколько пользователей одновременно изменяют одну и ту же строку, то в окончательном варианте строка будет иметь значение, определенное пользователем, последним изменившим запись. По сути, для *транзакции* не устанавливается никакой *блокировки*, которая гарантировала бы целостность данных. Для установки этого уровня используется команда:
  - SET TRANSACTION ISOLATION
  - LEVEL READ UNCOMMITTED
- READ COMMITTED – ***завершенное чтение***, при котором отсутствует черновое, *"грязное" чтение*. Тем не менее в процессе работы одной *транзакции* другая может быть успешно завершена и сделанные ею изменения зафиксированы. В итоге первая *транзакция* будет работать с другим набором данных. Это проблема *неповторяемого чтения*. Данный *уровень изоляции* установлен в SQL Server по умолчанию и устанавливается посредством команды:
  -
  - SET TRANSACTION ISOLATION
  - LEVEL READ COMMITTED
- REPEATABLE READ – ***повторяющееся чтение***. Повторное чтение строки возвратит первоначально считанные данные, несмотря на любые обновления, произведенные

другими пользователями до завершения *транзакции*. Тем не менее на этом *уровне изоляции* возможно возникновение *фантомов*. Его установка реализуется командой:

- SET TRANSACTION ISOLATION
- LEVEL REPEATABLE READ

- SERIALIZABLE – ***сериализуемость***. Чтение запрещено до завершения *транзакции*. Это максимальный *уровень изоляции*, который обеспечивает полную изоляцию *транзакций* друг от друга. Он устанавливается командой:
  - SET TRANSACTION ISOLATION
  - LEVEL SERIALIZABLE

В каждый момент времени возможен только один *уровень изоляции*.

Таблица 16.1. Уровень изоляции конкурирующей транзакции принят по умолчанию (READ COMMITTED). В примере шаги 4, 6 и 8 демонстрируют черновое чтение. Шаги 9 и 10 блокируются, потому что данные захвачены конкурирующей транзакцией.

Пользователь user1 Конкурирующая *транзакция*	**Пользователь user2 Текущая *транзакция***
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар
3.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4	4.SELECT * FROM Товар (читает измененные неподтвержденные данные)
5.DELETE FROM Товар WHERE КодТовара=4	6.SELECT * FROM Товар (читает измененные неподтвержденные данные)
7.INSERT Товар (Название, остаток) VALUES ('SS',999)	8.SELECT * FROM Товар (читает измененные неподтвержденные данные)
12.ROLLBACK TRANSACTION TRA	9.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется до окончания конкурирующей *транзакции*)
10.DELETE FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей *транзакции*)	11.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
13.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT	

Таблица 16.2. Уровень изоляции конкурирующей транзакции принят по умолчанию (READ COMMITTED). В примере шаги 4, 6 и 8 демонстрируют блокировку данных,

захваченных другой транзакцией, в то время как работа с другими данными разрешается (шаг 10).

Пользователь user1 Конкурирующая *транзакция*	**Пользователь user2 Текущая *транзакция***
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL READ COMMITTED BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар
3.UPDATE Товар SET остаток=остаток+10 (захватывает данные)	4.SELECT * FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей *транзакции*)
5.DELETE FROM Товар WHERE КодТовара=4	6.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется до окончания конкурирующей *транзакции*)
7.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (выполняется той *транзакцией*, которая первой захватила данные на изменение или удаление)	8.DELETE FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей *транзакции*)
9.INSERT Товар (Название, остаток) VALUES ('SS',999)	10.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
11.ROLLBACK TRANSACTION TRA SELECT @@TRANCOUNT	12.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

Таблица 16.3. Уровень изоляции конкурирующей транзакции принят по умолчанию (READ COMMITTED). На шаге 2 транзакция захватила данные чтением и блокирует работу с ними со стороны конкурирующей транзакции (шаги 3, 5), которая может лишь добавлять записи (шаг 7).

Пользователь user1 Конкурирующая *транзакция*	**Пользователь user2 Текущая *транзакция***
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ BEGIN TRANSACTION TRB

1.SELECT * FROM Товар	2.SELECT * FROM Товар (захватывает данные)
3.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется)	4.SELECT * FROM Товар (блокируется до окончания конкурирующей *транзакции*)
5.DELETE FROM Товар WHERE КодТовара=4 (блокируется)	6.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей *транзакцией*)
7.INSERT Товар (Название, остаток) VALUES ('SS',999) (выполняется)	8.DELETE FROM Товар WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей *транзакцией*)
10.ROLLBACK TRANSACTION TRA SELECT @@TRANCOUNT	9.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
	11.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

Таблица 16.4. Уровень изоляции конкурирующей транзакции принят по умолчанию (READ COMMITTED). Пример демонстрирует, что текущая транзакция захватила данные чтением (шаг 2) и блокирует любые действия с ними со стороны конкурирующей транзакции вплоть до вставки данных (шаг 7).

Пользователь user1 Конкурирующая *транзакция*	**Пользователь user2 Текущая *транзакция***
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар (захватывает данные)
3.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется)	4.SELECT * FROM Товар (выполняется)
5.DELETE FROM Товар WHERE КодТовара=4 (блокируется)	6.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей *транзакцией*)
7.INSERT Товар (наименование,	8.DELETE FROM Товар WHERE КодТовара=4

остаток) VALUES ('SS',999) (блокируется)	(выполняется, т.к. данные захвачены текущей *транзакцией*)
10.ROLLBACK TRANSACTION TRA SELECT @@TRANCOUNT	9.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
	11.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

Лекция: Основные методы защиты данных. Управление пользователями

1 Управление пользователями базы данных

Стабильная система *управления пользователями* – обязательное условие *безопасности данных*, хранящихся в любой реляционной СУБД. В языке SQL не существует единственной стандартной команды, предназначенной для *создания пользователей* базы данных – каждая реализация делает это по-своему. В одних реализациях эти специальные команды имеют определенное сходство, в то время как в других их синтаксис имеет существенные отличия. Однако независимо от конкретной реализации все основные принципы одинаковы.

2 Управление пользователями в среде MS SQL Server

Рассмотрим вопрос *создания пользователей* в среде MS SQL Server.
После проектирования логической структуры базы данных, связей между таблицами, ограничений целостности и других структур необходимо определить круг *пользователей*, которые будут иметь *доступ* к базе данных.
В системе SQL-сервер организована двухуровневая настройка ограничения *доступа* к данным. На первом уровне необходимо создать так называемую *учетную запись пользователя* (login), что позволяет ему подключиться к самому серверу, но не дает автоматического *доступа* к базам данных. На втором уровне для каждой базы данных SQL-сервера на основании *учетной записи* необходимо создать запись *пользователя*. На основе *прав*, выданных *пользователю* как *пользователю* базы данных (user), его регистрационное имя (login) получает *доступ* к соответствующей базе данных. В разных базах данных login одного и того же *пользователя* может иметь одинаковые или разные имена user с разными *правами доступа*. Иначе говоря, с помощью *учетной записи пользователя* осуществляется подключение к SQL-серверу, после чего определяются его уровни *доступа* для каждой базы данных в отдельности.
В системе SQL-сервер существуют дополнительные объекты – ***роли***, которые определяют уровень *доступа* к объектам SQL-сервера. Они разделены на две группы: назначаемые для *учетных записей пользователя* сервера и используемые для ограничения *доступа* к объектам базы данных.
Итак, на уровне сервера система безопасности оперирует следующими понятиями:

- *аутентификация*;
- *учетная запись*;
- встроенные *роли* сервера.

На уровне базы данных применяются следующие понятия;

- *пользователь* базы данных;
- фиксированная *роль* базы данных;
- *пользовательская роль* базы данных.

Режимы аутентификации

SQL Server предлагает два режима *аутентификации пользователей*:

- режим *аутентификации* средствами Windows NT/2000;
- смешанный режим *аутентификации* (Windows NT Authentication and SQL Server Authentication).

Администрирование системы безопасности

Для ***создания пользователя*** в среде MS SQL Server следует предпринять следующие шаги:

1. Создать в базе данных *учетную запись пользователя*, указав для него *пароль* и принятое по умолчанию имя базы данных (процедура sp_addlogin).
2. Добавить этого *пользователя* во все необходимые базы данных (процедура sp_adduser).
3. Предоставить ему в каждой базе данных соответствующие *привилегии* (команда GRANT) .

Создание новой учетной записи может быть произведено с помощью системной хранимой процедуры:

```
sp_addlogin
[@login=] 'учетная_запись'
[, [@password=] 'пароль']
[, [@defdb=] 'база_данных_по_умолчанию']
```

После завершения *аутентификации* и получения *идентификатора учетной записи (login ID)* *пользователь* считается зарегистрированным, и ему предоставляется *доступ* к серверу. Для каждой базы данных, к объектам которой он намерен получить *доступ*, *учетная запись пользователя* (login) ассоциируется с *пользователем* (user) конкретной базы данных, что осуществляется посредством процедуры:

```
sp_adduser
[@loginame=] 'учетная_запись'
[, [@name_in_db=] 'имя_пользователя']
[, [@grpname=] 'имя_роли']
```

Отобразить учетную запись Windows NT в имя *пользователя* позволяет хранимая процедура:

```
sp_grantdbaccess
[@login=] ‘учетная_запись’
[, [@name_in_db=]‘имя_пользователя’]
```

Пользователь, который создает объект в базе данных (таблицу, хранимую процедуру, просмотр), становится его ***владельцем***. *Владелец объекта* (database object owner dbo) имеет все *права доступа* к созданному им объекту. Чтобы *пользователь* мог создать объект, *владелец* базы данных (dbo) должен предоставить ему соответствующие *права*. Полное имя создаваемого объекта включает в себя имя создавшего его *пользователя*.

Владелец объекта не имеет специального *пароля* или особых *прав доступа*. Он неявно имеет полный *доступ*, но должен явно предоставить *доступ* другим *пользователям*.

SQL Server позволяет передавать *права* владения от одного *пользователя* другому с помощью процедуры:

```
sp_changeobjectowner
[@objname=] ‘имя_объекта’
[@newowner=] ‘имя_владельца’
```

Роль позволяет объединить в одну группу *пользователей*, выполняющих одинаковые функции. В SQL Server реализовано два вида стандартных *ролей*: на уровне сервера и на уровне баз данных. При установке SQL Server создаются фиксированные *роли* сервера (например, sysadmin с *правом* выполнения любых функций SQL-сервера) и фиксированные *роли* базы данных

(например, db_owner с *правом* полного *доступа* к базе данных или db_accessadmin с *правом* добавления и удаления *пользователей*). Среди фиксированных *ролей* базы данных существует *роль* public, которая имеет специальное назначение, поскольку ее членами являются все *пользователи*, имеющие *доступ* к базе данных.
Можно включить любую *учетную запись* SQL Server (login) или *учетную запись* Windows NT в любую *роль* сервера.
Роли базы данных позволяют объединять *пользователей* в одну административную единицу и работать с ней как с обычным *пользователем*. Можно назначить *права доступа* к объектам базы данных для конкретной *роли*, при этом автоматически все члены этой *роли* наделяются одинаковыми *правами*.
В *роль* базы данных можно включить *пользователей* SQL Server, *роли* SQL Server, *пользователей* Windows NT.
Различные действия по отношению к *роли* осуществляются при помощи специальных процедур:

- создание новой *роли*:
  - sp_addrole
  - [@rolename=] 'имя_роли'
  - [, [@ownername=] 'имя_владельца']
- добавление *пользователя* к *роли*:
  - sp_addrolemember
  - [@rolename=] 'имя_роли',
  - [@membername=] 'имя_пользователя'
- удаление *пользователя* из *роли*:
  - sp_droprolemember
  - [@rolename=] 'имя_роли',
  - [@membername=] 'имя_пользователя'
- удаление *роли*:
  - sp_droprole
  - [@rolename=] 'имя_роли'

3 Управление доступом к данным

Определение привилегий в стандарте языка

Каждая СУБД должна поддерживать механизм, гарантирующий, что *доступ* к базе данных смогут получить только те *пользователи*, которые имеют соответствующее разрешение. Язык SQL включает операторы GRANT и REVOKE, предназначенные для организации защиты таблиц в базе данных. Механизм защиты построен на использовании *идентификаторов пользователей*, предоставляемых им *прав* владения и *привилегий*.
Идентификатором пользователя называется обычный идентификатор языка SQL, применяемый для обозначения некоторого *пользователя* базы данных. Каждому *пользователю* должен быть назначен собственный *идентификатор*, присваиваемый администратором базы данных. Из очевидных соображений безопасности *идентификатор пользователя*, как *правило*, связывается с некоторым *паролем*. Каждый выполняемый СУБД SQL-оператор выполняется от имени какого-либо *пользователя*. *Идентификатор пользователя* определяет, на какие объекты базы данных *пользователь* может ссылаться и какие операции с этими объектами он имеет *право* выполнять.
Каждый созданный в среде SQL объект имеет своего *владельца*, который изначально является единственной персоной, знающей о существовании данного объекта и имеет *право* выполнять с ним любые операции.

Привилегиями, или ***правами***, называются действия, которые *пользователь* имеет *право* выполнять в отношении данной таблицы базы данных или представления. В стандарте SQL определяется следующий набор *привилегий*:

- SELECT – *право* выбирать данные из таблицы;
- INSERT – *право* вставлять в таблицу новые строки;
- UPDATE – *право* изменять данные в таблице;
- DELETE – *право* удалять строки из таблицы;
- REFERENCES – *право* ссылаться на столбцы указанной таблицы в описаниях требований поддержки целостности данных;
- USAGE – *право* использовать домены, проверки и наборы символов.

Привилегии INSERT и UPDATE могут ограничиваться лишь отдельными столбцами таблицы, в этом случае *пользователю* разрешается модифицировать значения только указанных столбцов. Аналогичным образом *привилегия* REFERENCES может распространяться исключительно на отдельные столбцы таблицы, что позволит использовать их имена в формулировках требований защиты целостности данных – например, в предложениях CHECK и FOREIGN KEY, входящих в определение других таблиц, тогда как применение для подобных целей остальных столбцов будет *запрещено*.
Когда *пользователь* с помощью оператора CREATE TABLE создает новую таблицу, он автоматически становится ее *владельцем* и получает по отношению к ней полный набор *привилегий*, которых остальные *пользователи* исходно не имеют. Чтобы обеспечить им *доступ*, *владелец* должен явным образом предоставить необходимые *права*, для чего используется оператор GRANT.
Создавая представление с помощью оператора CREATE VIEW, *пользователь* автоматически становится *владельцем* этого представления и также получает полный набор *прав*. Для создания представления *пользователю* достаточно иметь *привилегию* SELECT для всех входящих в него таблиц и *привилегию* REFERENCES для всех столбцов, упоминаемых в определении этого представления. *Привилегии* INSERT, UPDATE и DELETE в отношении созданного представления *пользователь* получит только в том случае, если имеет соответствующие *привилегии* в отношении всех используемых в представлении таблиц.

Предоставление привилегий пользователям

Оператор GRANT применяется для ***предоставления привилегий*** в отношении поименованных объектов базы данных указанным *пользователям*. Обычно его использует *владелец* таблицы с целью *предоставления доступа* к ней другим *пользователям*. Оператор GRANT имеет следующий формат:

```
<предоставление_привилегий>::=
GRANT {<привилегия>[,...n] |
   ALL PRIVILEGES}
ON имя_объекта
TO {<идентификатор_пользователя>
   [,...n]| PUBLIC}
[ WITH GRANT OPTION]
```

Параметр <привилегия> представляет собой:

```
<привилегия>::=
{SELECT | DELETE | INSERT
   [(имя_столбца[,...n])]
| UPDATE [(имя_столбца[,...n])]}
| REFERENCES [(имя_столбца[,...n])] |
   USAGE }
```

Из соображений упрощения в операторе GRANT можно указать ключевое слово ALL PRIVILEGES, что позволит предоставить указанному *пользователю* все существующие *привилегии* без необходимости их перечисления. Кроме того, в этом операторе может указываться ключевое слово PUBLIC, означающее *предоставление доступа* указанного типа не только всем существующим *пользователям*, но также и всем тем, кто будет определен в базе данных впоследствии.
Параметр имя_объекта может использоваться как имя таблицы базы данных, представления, домена, набора символов, проверки.
Благодаря параметру WITH GRANT OPTION, указанные в операторе GRANT *пользователи* имеют *право* передавать все *предоставленные* им в отношении указанного объекта *привилегии* другим *пользователям*, которые, в свою очередь, будут наделены точно таким же *правом* передачи своих полномочий. Если данный параметр не будет указан, получатель *привилегии* не сможет передать свои *права* другим *пользователям*. Таким образом, *владелец* объекта может четко контролировать, кто получил *право доступа* к объекту и какие полномочия ему *предоставлены*.

Отмена предоставленных пользователям привилегий

В языке SQL для ***отмены привилегий***, *предоставленных пользователям* посредством оператора GRANT, используется оператор REVOKE. С помощью этого оператора могут быть *отменены* все или некоторые из *привилегий*, полученных указанным *пользователем* раньше.
Оператор REVOKE имеет следующий формат:

```
<отмена_привилегий>::=
REVOKE[GRANT OPTION FOR]
    {<привилегия>[,...n]
    | ALL PRIVILEGES}
ON имя_объекта
FROM {<идентификатор_пользователя>
    [,...n]| PUBLIC}
[RESTRICT | CASCADE]
```

Ключевое слово ALL PRIVILEGES означает, что для указанного *пользователя отменяются* все *привилегии*, *предоставленные* ему ранее тем пользователем, который ввел данный оператор.
Необязательная фраза GRANT OPTION FOR позволяет для всех *привилегий*, переданных в исходном операторе GRANT фразой WITH GRANT OPTION, отменять возможность их передачи независимо от самих *привилегий*.
Если в операторе указано ключевое слово RESTRICT, успешное выполнение команды REVOKE возможно лишь в том случае, когда перечисленные в операторе *привилегии* не могут послужить причиной появления у каких-либо других *пользователей* так называемых "оставленных" *привилегий*. С помощью параметра CASCADE удаляются все *привилегии*, которые иначе могли бы остаться у других *пользователей*.
"Оставленными" являются *привилегии*, сохранившиеся у *пользователя*, которому они в свое время были *предоставлены* с помощью параметра GRANT OPTION.
Поскольку наличие *привилегии* необходимо для создания определенных объектов, вместе с ее удалением можно лишиться *права*, за счет использования которого был образован тот или иной объект (подобные объекты называются "брошенными"). Если в результате выполнения оператора REVOKE могут появиться брошенные объекты (например, представления), право будет отменено при условии, что в нем не указывается ключевое слово CASCADE. Если ключевое слово CASCADE в операторе присутствует, то для любых брошенных объектов, возникающих при выполнении исходного оператора REVOKE, будут автоматически выданы операторы DROP.
Привилегии, которые были *предоставлены* указанному *пользователю* другими *пользователями*, не могут быть затронуты оператором REVOKE. Следовательно, если другой *пользователь*

также предоставил данному *пользователю* удаляемую *привилегию*, то *право доступа* к соответствующей таблице у указанного *пользователя* сохранится. Например, пусть *пользователь* A и *пользователь* E имели *право* INSERT на таблицу Товар. *Пользователь* A предоставил *пользователю* B *привилегию* INSERT для таблицы Товар, причем с указанием WITH GRANT OPTION (этап 1). *Пользователь* B передал эту *привилегию пользователю* C (этап 2). Затем *пользователь* C получил ее же от *пользователя* E (этап 3). Далее *пользователь* C предоставил упомянутую *привилегию пользователю* D (этап 4). Когда *пользователь* A *отменяет привилегию* INSERT для *пользователя* B, она не может быть *отменена* и для *пользователя* C, поскольку ранее он уже получил ее от *пользователя* E. Если бы *пользователь* E не предоставил данной *привилегии пользователю* C, то удаление *привилегии пользователя* B имело бы следствием каскадное удаление *привилегий* для *пользователей* C и D (см. таблицу 17.1).

4 Реализация прав на доступ к объектам баз данных в среде MS SQL Server

Категории прав в среде MS SQL Server

При подключении к SQL Server все возможные действия *пользователей* определяются *правами* (*привилегиями*, разрешениями), выданными их *учетной записи*, группе или *роли*, в которых они состоят.
Права можно разделить на три категории:

- *права на доступ к объектам*;
- *права на выполнение команд*;
- *неявные права*.

Таблица 17.1.

Пользователь A	*Пользователь* B	*Пользователь* C	*Пользователь* D	*Пользователь* E
GRANT INSERT ON Товар TO B WITH GRANT OPTION	Получение *права*			
	GRANT INSERT ON Товар TO C WITH GRANT OPTION	Получение *права* от B. Получение *права* от E		GRANT INSERT ON Товар TO C WITH GRANT OPTION
		GRANT INSERT ON Товар TO D	Получение *права*	
REVOKE INSERT ON Товар TO B CASCADE	Отмена *права*	Сохранение *права*	Сохранение *права*	Сохранение *права*

Работа с данными и выполнение хранимых процедур требуют наличия класса *доступа*, называемого *правами на доступ к объектам* баз данных. Под объектами подразумеваются таблицы, столбцы таблиц, представления, хранимые процедуры.
Для различных объектов применяются разные наборы *прав доступа* к ним:

- SELECT, INSERT, UPDATE, DELETE, REFERENCES – для таблицы или представления;
- SELECT, UPDATE – для конкретного столбца таблицы или представления;
- EXECUTE – для хранимых процедур и функций.

Право INSERT позволяет вставлять новые строки в таблицу или представление и выдается только на уровне таблицы или представления; оно не может быть выдано на уровне столбца.
Право UPDATE выдается либо на уровне таблицы, что позволяет изменять в ней все данные, либо на уровне отдельного столбца, что разрешает изменять данные только в его пределах.
Право DELETE позволяет удалять строки из таблицы или представления, выдается только на уровне таблицы или представления, но не может быть выдано на уровне столбца.
Право SELECT разрешает выборку данных и может выдаваться как на уровне таблицы, так и на уровне отдельного столбца.
Право REFERENCES предоставляет возможность ссылаться на указанный объект. Применительно к таблицам разрешает создавать внешние ключи, ссылающиеся на первичный ключ или уникальный столбец этой таблицы.

Предоставление прав

Для *управления разрешением пользователя на доступ к объектам* базы данных используется команда:

```
<предоставление_привилегий>::=
GRANT { ALL [ PRIVILEGES] | <привилегия>
   [,...n]}
{ [(    имя_столбца [,...n])]
   ON { имя_таблицы |
      имя_просмотра} |
         ON {имя_таблицы |
      имя_просмотра }
         ([имя_столбца
            [,...n])]
   | ON {имя_хранимой_процедуры |
      имя_внешней_процедуры}}
TO {    имя_пользователя | имя_группы |
      имя_роли} [,...n]
[WITH GRANT OPTION ]
[AS {имя_группы | имя_роли }]
```

Параметр <привилегия> представляет собой следующую конструкцию:

```
<привилегия>::=
{SELECT | DELETE | INSERT |
 UPDATE | EXECUTE | REFERENCES }
```

Параметр WITH GRANT OPTION поможет *пользователю*, которому вы предоставляете *права*, назначить *права* на *доступ* к объекту другим *пользователям*. Его использование требует особой осторожности, поскольку при этом *владелец* теряет контроль над *предоставлением прав* на *доступ* другим *пользователям*. Лучше всего ограничить круг *пользователей*, обладающих возможностью *управлять назначением прав*.
Необязательный параметр AS {имя_группы | имя_роли } позволяет указать участие *пользователя* в *роли*, обеспечивающей *предоставление прав* другим *пользователям*.

Единственное *право доступа*, которое может быть *предоставлено* для хранимой процедуры, – *право* на ее выполнение (EXECUTE). Естественно, кроме этого *владелец* хранимой процедуры может просматривать и изменять ее код.
Для функции можно выдать *право* на ее выполнение, а кроме того, выдать *право* REFERENCES, что обеспечит возможность связывания функции с объектами, на которые она ссылается. Такое связывание позволит запретить внесение изменений в структуру объектов, способных привести к нарушению работы функции.

Права на выполнение команд SQL

Этот класс *прав* контролирует возможность создания объектов в базе данных, самой базы данных и выполнения процедуры резервного копирования. Можно использовать следующую команду для *предоставления права на выполнение команд* SQL:

```
<предоставление_права_выполнения>::=
GRANT {ALL | <команда>[,...n]}
TO {имя_пользователя | имя_группы |
   имя_роли} [,...n]
```

Параметр <команда> представляет собой следующую конструкцию:

```
<команда>::=
{CREATE DATABASE | CREATE TABLE |
   CREATE VIEW | CREATE DEFAULT |
   CREATE RULE | CREATE PROCEDURE
      | BACKUP DATABASE |
BACKUP LOG | ALL }
```

Таким образом, можно предоставить *право* на создание базы данных, таблицы, просмотра, умолчания, правила, хранимой процедуры, резервной копии базы данных и журнала транзакций или предоставить сразу все вышеперечисленные *права*.

Неявные права

Выполнение некоторых действий не требует явного разрешения и *доступно* по умолчанию. Эти действия могут быть выполнены только членами *ролей* сервера или *владельцами* объектов в базе данных.
Неявные права не предоставляются *пользователю* непосредственно, он получает их лишь при определенных обстоятельствах. Например, *пользователь* может стать *владельцем* объекта базы данных, только если сам создаст объект либо если кто-то другой передаст ему *право* владения своим объектом. Таким образом *владелец* объекта автоматически получит *права* на выполнение любых действий с объектом, в том числе и на *предоставление доступа* к объекту другим *пользователям*. Эти *права* нигде не указываются, выполнять любые действия позволяет только факт владения объектом.

Запрещение доступа

Система безопасности SQL Server имеет иерархическую структуру, и поэтому *роли* базы данных включают в себя *учетные записи* и группы Windows NT, *пользователей* и *роли* SQL Server. *Пользователь* же, в свою очередь, может участвовать в нескольких *ролях* и одновременно иметь разные *права доступа* для разных *ролей*. Когда одна из *ролей*, в которых состоит *пользователь*, имеет разрешение на *доступ* к данным, он автоматически имеет аналогичные *права*. Тем не менее, если возникает необходимость, *пользователю* можно запретить *доступ* к данным или командам, тогда аннулируются все разрешения на *доступ*, полученные им на любом уровне иерархии. При этом гарантируется, что *доступ* останется *запрещенным* независимо от разрешений, предоставленных на более высоком уровне.
Для ***запрещения доступа*** к объектам базы данных используется команда:

```
<запрещение_доступа>::=
```

DENY {ALL [PRIVILEGES]| | <привилегия>
[,...n]}
{ [(имя_столбца [,...n])]
ON { имя_таблицы |
имя_просмотра}
| ON {имя_таблицы | имя_просмотра }
[имя_столбца [,...n])]
| ON {имя_хранимой_процедуры |
имя_внешней_процедуры}}
TO {имя_пользователя | имя_группы |
имя_роли}
[,...n]
[CASCADE]

Параметр CASCADE позволяет отзывать *права* не только у конкретного *пользователя*, но также и у всех тех, кому он предоставил аналогичные *права*.

Для *запрещения* выполнения команд SQL применяется оператор:

<запрещение_выполнения>::=
DENY {ALL | <команда>[,...n]}
TO {имя_пользователя | имя_группы |
имя_роли} [,...n]

Неявное отклонение доступа

Неявное отклонение подобно *запрещению доступа* с тем отличием, что оно действует только на том уровне, на котором определено. Если *пользователю* на определенном уровне *неявно отклонен доступ*, он все же может получить его на другом уровне иерархии через членство в *роли*, имеющей *право* просмотра. По умолчанию *доступ пользователя* к данным *неявно отклонен*. Для *неявного отклонения доступа* к объектам базы данных используется команда:

<неявное_отклонение_доступа>::=
REVOKE [GRANT OPTION FOR]
{ALL [PRIVILEGES]| | <привилегия>
[,...n]}
{ [(имя_столбца [,...n])] ON
{ имя_таблицы | имя_просмотра}
| ON {имя_таблицы |
имя_просмотра }
[имя_столбца [,...n])]
| ON {имя_хранимой_процедуры |
имя_внешней_процедуры}}
TO | FROM {имя_пользователя |
имя_группы |
имя_роли}[,...n]
[CASCADE]
[AS {имя_группы | имя_роли }]

Для *неявного отклонения разрешения на выполнение команд* SQL используется следующая команда:

<неявное_отклонение_разрешения>::=
REVOKE {ALL | <команда>[,...n]}
FROM {имя_пользователя | имя_группы |
имя_роли}[,...n]

Смысл параметров аналогичен параметрам команд GRANT и DENY. Параметр GRANT OPTION FOR используется, когда необходимо отозвать *право*, *предоставленное* параметром WITH GRANT OPTION команды GRANT. *Пользователь* сохраняет разрешение на *доступ* к объекту, но теряет возможность предоставлять это разрешение другим *пользователям*.

Конфликты доступа

Разрешения, *предоставленные роли* или группе, наследуются их членами. Хотя *пользователю* может быть *предоставлен доступ* через членство в одной *роли*, *роль* другого уровня может иметь *запрещение* на действие с объектом. В таком случае возникает ***конфликт доступа***.
При разрешении *конфликтов доступа* SQL Server руководствуется следующим принципом: разрешение на *предоставление доступа* имеет самый низкий приоритет, а на *запрещение доступа* – самый высокий. Это значит, что *доступ* к данным может быть получен только явным его *предоставлением* при отсутствии *запрещения доступа* на любом другом уровне иерархии системы безопасности. Если *доступ* явно не *предоставлен*, *пользователь* не сможет работать с данными.

Пример 17.1. Создать новую базу данных, нового пользователя для этой базы данных, предоставив ему все права.

```
-- создание администратором новой
-- базы данных
CREATE DATABASE basa_user
-- создание нового пользователя с
-- именем UserA и паролем ‘123’
-- базой данных по умолчанию для
-- пользователя UserA будет база
-- с именем basa_user.
sp_addlogin 'UserA','123','basa_user'
-- переход в базу данных basa_user
USE basa_user
-- добавление в текущую базу данных
-- (basa_user) пользователя с именем
-- userA
sp_adduser 'UserA'
-- предоставление пользователю userA
-- в базе данных basa_user всех прав
GRANT ALL TO UserA
```

Пример 17.1. Создание новой базы данных, нового пользователя для этой базы данных, с предоставлением ему всех прав. (html, txt)

Пример 17.2. Использование ролей.
Создадим *роль* stud и включим в эту *роль* двух *пользователей* user1 и user2:

```
sp_addrole 'stud'
sp_addrolemember 'stud','user1'
sp_addrolemember 'stud','user2'
```

Предоставим *права роли* stud и непосредственно *пользователю* user2:

```
GRANT SELECT, INSERT ON Товар TO stud
GRANT SELECT, INSERT ON Товар TO user2
```

После выполнения этих команд *пользователи* user1 и user2 могут выполнять команды выборки и добавления записи в таблицу Товар.
Приостановим *право* на выполнение вставки в таблицу Товар для *роли* stud:

```
REVOKE INSERT ON Товар TO stud
```

После выполнения предыдущей команды *пользователь* user1 теряет *право* вставки записи, а user2 сохраняет это *право*, поскольку *право* вставки *предоставлено* ему явно.
Выполним команду

```
DENY INSERT ON Товар TO stud.
```

После выполнения этой команды оба *пользователя* лишаются *права* вставки в таблицу Товар.

2. Лабораторная часть

Министерство образования республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

В.Н. ЛЕВАНЦОВ

БАЗЫ И БАНКИ ДАННЫХ

Практическое пособие
по выполнению лабораторных работ

часть1

Гомель 2012

Министерство образования республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

В.Н. ЛЕВАНЦОВ

БАЗЫ И БАНКИ ДАННЫХ

Практическое пособие для студентов специальности I – 53 01 02
«Автоматизированные системы обработки информации»

Гомель 2012

Авторы-составители:

Леванцов Виктор Николаевич

Практическое пособие по выполнению лабораторных работ по курсу «Базы и банки данных» адресовано студентам специальности I – 53 01 02 «АСОИ» и включает краткие теоретические сведения по основным темам курса, требования по выполнению и оформлению практических заданий и содержанию отчета.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Широкое распространение компьютерной техники и вычислительных сетей привело к увеличению объемов информации, хранимой в электронном виде, что повлекло необходимость усовершенствования принципов предварительной обработки и структурирования вводимых данных, а также алгоритмов доступа и модификации данных. Это привело к концепции баз данных и систем управления базами данных (СУБД). В настоящее время наиболее распространенными и промышленно применимыми являются реляционные базы данных. Это позволило ускорить процесс обработки информации и уменьшить время от момента возникновения данных до момента принятия решения. Однако следствием этого стала потребность в унификации доступа к данным распределенных систем, уменьшению времени получения локальными пользователям запрошенной информации. В настоящее время основным решением этой проблемы является использование языка SQL и SQL-запросов для организации работы в сетевых СУБД. Правильный выбор структуры и принципов организации данных определяет эффективность таких систем, возможности по анализу накопленных данных.

Необходимость изучения: изучение дисциплины «Базы и банки данных» предусмотрено учебным планом подготовки специалистов специальности I-53 01 02.

Цель изучения дисциплины «Базы и банки данных»– обучение эффективным принципам организации хранения и доступа к данным, баз данных и систем управления базами данных.

Выполнение практических работ по дисциплине «Базы и банки данных» позволяет студентам закрепить теоретические знания по предмету и приобрести практические навыки по проектированию баз данных и применения систем управления базами данных для обработки информации. Каждая тема содержит краткие теоретические сведения, необходимые для выполнения предлагаемых практических заданий, с иллюстрациями и примерами. Для каждого задания предусмотрены варианты для выполнения.

ТЕМА 1 ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Основные понятия по теме

1 Базы и банки данных, системы управления базами данных
2 Реляционные базы данных
3 Базовые понятия реляционных баз данных
4 Проектирование баз данных, нормализация отношений

Восприятие реального мира можно соотнести с последовательностью разных, хотя иногда и взаимосвязанных, явлений. С давних времен люди пытались описать эти явления (даже тогда, когда не могли их понять). Такое описание называют данными.

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 - базы данных (БД) и банки данных.

База данных— совокупность экземпляров различных типов записей и отношений между записями, агрегатами данных, элементами данных.

Система баз данных - в большинстве систем термин *база данных* относится не ко *всем* типам записей, а только к некоторой их совокупности. В одной системе может использоваться несколько баз данных, однако предполагается, что различные базы данных разделены и не связаны между собой. Для обозначения совокупности баз данных требуется некоторый термин —*система баз данных.*

Иногда для обозначения совокупности баз данных употребляется термин *банк данных.* Некоторые специалисты вместо *базы* данных используют термин *банк данных,* а под базой данных подразумевают совокупность банков данных.

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.

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 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 распространенными являются СУБД, основанные на реляционной модели данных.

Реляционные системы далеко не сразу получили широкое распространение. В то время, как основные теоретические результаты в этой области были получены еще в 70-х, и тогда же появились первые прототипы реляционных СУБД, долгое время считалось невозможным добиться эффективной реализации таких систем. Однако их преимущества и постепенное накопление методов и алгоритмов организации реляционных баз данных и управления ими привели к тому, что уже в середине 80-х годов реляционные системы практически вытеснили с мирового рынка ранние СУБД.

В реляционной базе данные организованы в виде таблиц.

Реляционная модель отличается от иерархической и сетевой модели простым и единообразным представлением данных. Основной принцип - использование логических операций над таблицами с целью извлечения из таблицы желаемого отношения и формирования новых таблиц.

Основные свойства: отсутствуют одинаковые строки, порядок строк не существенен, порядок столбцов не существенен, все значения нельзя разбить без потери информации.

Достоинства **реляционных баз данных:**

- упрощение схематических данных для пользователя;
- улучшение логической и физической независимости;
- обеспечение пользователя языком высокого уровня;

- оптимизация доступа к данным, улучшение целостности и защиты данных;
- возможность различных применений.

Примерами реляционных БД могут быть: FoxPro, Clipper, MS Access.

Основными понятиями реляционных баз данных являются тип данных, домен, атрибут, кортеж, первичный ключ и отношение.

Тип *данных*

Понятие тип данных в реляционной модели данных полностью адекватно понятию типа данных в языках программирования. Обычно в современных реляционных БД допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как "деньги"), а также специальных "темпоральных" данных (дата, время, временной интервал). Достаточно активно развивается подход к расширению возможностей реляционных систем абстрактными типами данных.

Домен

Понятие домена более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат "истина", то элемент данных является элементом домена.

Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа.

Например, домен "Имена" может быть определен на базовом типе строк символов, но в число его значений могут входить только те строки, которые могут изображать имя (в частности, такие строки не могут начинаться с мягкого знака).

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену.

В большинстве реляционных СУБД понятие домена не используется, хотя, например, в Oracle V.7 оно уже поддерживается.

Схема *отношения, схема базы данных*

Схема отношения - это именованное множество пар

{имя атрибута, имя домена/типа}

Степень или "арность" схемы отношения - мощность этого множества.

Схема БД (в структурном смысле) - это набор именованных схем отношений.

Кортеж, отношение

Кортеж, соответствующий данной схеме отношения, - это множество пар
{имя атрибута, значение},
которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа

данных, если понятие домена не поддерживается). Тем самым, степень или "арность" кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения. Попросту говоря, кортеж - это набор именованных значений заданного типа.

Отношение - это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отношение-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения. На самом деле, понятие схемы отношения ближе всего к понятию структурного типа данных в языках программирования.

В классических реляционных базах данных после определения схемы базы данных изменяются только отношения-экземпляры. В них могут появляться новые и удаляться или модифицироваться существующие кортежи. Однако во многих реализациях допускается и изменение схемы базы данных: определение новых и изменение существующих схем отношения. Это принято называть *эволюцией схемы базы данных***.**

Обычным житейским представлением отношения является таблица, заголовком которой является схема отношения, а строками - кортежи отношения-экземпляра; в этом случае имена атрибутов именуют столбцы этой таблицы. Поэтому иногда говорят "столбец таблицы", имея в виду "атрибут отношения".

Реляционная база данных - это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

Как видно, основные структурные понятия реляционной модели данных (если не считать понятия домена) имеют очень простую интуитивную интерпретацию, хотя в теории реляционных БД все они определяются абсолютно формально и точно.

Соответствие базовых понятий реляционных БД обычному графическому представлению таблиц приведено на рисунке 1.1.

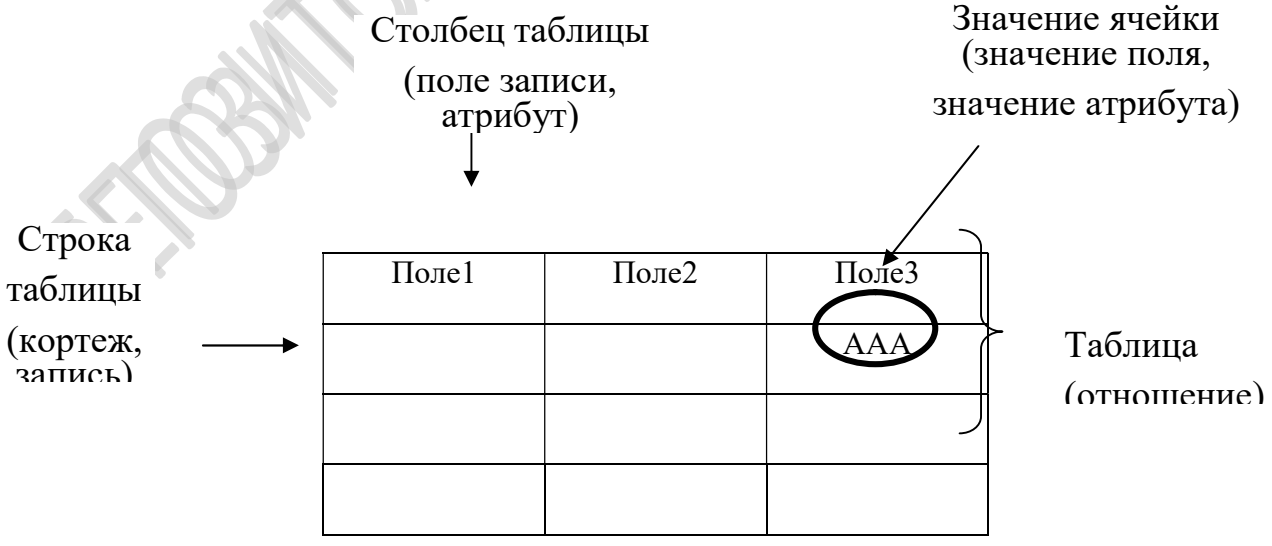


Рисунок 1.1 – Таблица реляционной базы данных

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. 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 администратор базы данных (АБД), который далее занимается детальным проектированием базы.

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.

Таким образом, проектирование реляционной БД заключается в разработке структуры данных, т.е. в определении состава таблиц и связей между ними. При этом структура должна быть эффективной и обеспечивать:

- быстрый доступ к данным;
- отсутствие дублирования (повторения) данных
- целостность данных.

Формальным методом для получения базы данных, структура которой будет удовлетворять приведенным выше требованиям, является *нормализация*.

Нормализация – это разбиение таблицы на две или более, обладающих лучшими свойствами при включении, изменении и удалении данных. Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором *каждый факт появляется лишь в одном месте*, т.е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных.

В теории реляционных баз данных обычно выделяется следующая **последовательность нормальных форм:**

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

Каждая таблица в реляционной БД удовлетворяет условию, в соответствии с которым в позиции на пересечении каждой строки и столбца таблицы всегда находится единственное атомарное значение, и никогда не может быть множества таких значений. Любая таблица, удовлетворяющая этому условию, называется нормализованной. Фактически, ненормализованные таблицы, т.е. таблицы, содержащие повторяющиеся группы, даже не допускаются в реляционной БД.

Для определения нормальных форм предварительно необходимо ввести следующие определения:

Первичный ключ:

атрибут (или набор атрибутов), который может быть использован для однозначной идентификации конкретного кортежа (строки, записи), называется *первичным ключом*.

Первичный ключ не должен иметь дополнительных атрибутов. Это значит, что если из первичного ключа исключить произвольный атрибут, оставшихся атрибутов будет недостаточно для однозначной идентификации отдельных кортежей.

Внешний ключ:

для поддержания ссылочной целостности данных во многих СУБД имеется механизм так называемых *внешних ключей*. Смысл этого механизма состоит в том, что некоему атрибуту (или группе атрибутов) одного отношения назначается ссылка на первичный ключ другого отношения; тем самым закрепляются связи подчиненности между этими отношениями. При этом отношение, на первичный ключ которого ссылается внешний ключ другого отношения, называется *master-отношением*, или главным отношением; а отношение, от которого исходит ссылка, называется *detail-отношением*, или подчиненным отношением.

Таблица находится в **первой нормальной форме (1НФ)** тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто.

Теория нормализации основывается на наличии той или иной зависимости между полями таблицы.

Функциональная зависимость. Поле B таблицы функционально зависит от поля A той же таблицы в том и только в том случае, когда в любой заданный момент времени для каждого из различных значений поля A обязательно существует только одно из различных значений поля B. Отметим, что здесь допускается, что поля A и B могут быть составными.

Например, рассмотрим таблицу «Поставщики» на рисунке 1.2:

ПОСТАВЩИКИ

Поставщик	**Город**	**Страна**

"Полесье"	**Киев**	**Украина**
"Наталка"	Киев	Украина
"Хуанхэ"	Пекин	Китай
“Юрмала”	Рига	Латвия
"Лайма"	Рига	Латвия
...	...	...

Рисунок 1.2 –Таблица реляционной БД с функциональной зависимостью

В приведенной таблице «Поставщики» поле «Страна» функционально зависит от составного ключа («Поставщик», «Город»). Однако последняя зависимость не является функционально полной, так как «Страна» функционально зависит и от части ключа – поля «Город».

Полная функциональная зависимость. Поле B находится в полной функциональной зависимости от составного поля A, если оно функционально зависит от A и не зависит функционально от любого подмножества поля A.

Многозначная зависимость. Поле A многозначно определяет поле B той же таблицы, если для каждого значения поля A существует хорошо определенное множество соответствующих значений B.

ОБУЧЕНИЕ

Дисциплина	**Преподаватель**	**Учебник**
Информатика	Шипилов П.А.	Форсайт Р. Паскаль для всех
Информатика	Шипилов П.А.	Уэйт М. и др. Язык Си
Информатика	Голованевский Г.Л.	Форсайт Р. Паскаль для всех
Информатика	Голованевский Г.Л.	Уэйт М. и др. Язык Си
...	...	...

Рисунок 1.3 – Таблица с многозначной зависимостью

Для примера рассмотрим таблицу "Обучение", приведенную на рисунке 1.3. В ней есть многозначная зависимость "Дисциплина-Преподаватель": дисциплина (в примере Информатика)

может читаться несколькими преподавателями (в примере Шипиловым и Голованевским). Есть и другая многозначная зависимость "Дисциплина-Учебник": при изучении Информатики используются учебники "Паскаль для всех" и "Язык Си". При этом Преподаватель и Учебник не связны функциональной зависимостью, что приводит к появлению избыточности (для добавление еще одного учебника придется ввести в таблицу две новых строки). Дело улучшается при замене этой таблицы на две: (Дисциплина-Преподаватель и Дисциплина-Учебник).

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа (в частности, первичного).

Взаимно независимые атрибуты: два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

При классическом подходе к проектированию базы данных весь процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами. Процесс проектирования представляет собой процесс нормализации схем отношений.

Основной формой считается 3НФ – для большинства баз данных она является достаточной для обеспечения целей проектирования.

1НФ - таблица находится в первой нормальной форме (1НФ) тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто. (Любое поле таблицы содержит неделимую информацию и в таблице определен первичный ключ).

2НФ - Таблица находится во второй нормальной форме (2NF) в том и только в том случае**, когда находится в 1NF, и каждый ее неключевой атрибут полностью зависит от первичного ключа.**

3НФ – Таблица находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый неключевой атрибут нетранзитивно **зависит от первичного ключа.** (Иными словами, таблица должна находиться во второй нормальной форме и ни одно из ее неключевых полей не должно однозначно идентифицироваться значением другого неключевого поля (полей)).

Теоретики реляционных систем Кодд и Бойс обосновали и предложили более строгое определение для 3НФ, которое учитывает, что в таблице может быть несколько возможных ключей. Таблица, соответствующая этому определению называется таблицей *в улучшенной третьей форме* или таблицей в *нормальной форме Бойса-Кодда*.

Таблица находится в **нормальной форме Бойса-Кодда** (НФБК), если и только если любая функциональная зависимость между его полями сводится к полной функциональной зависимости от возможного ключа.

В следующих нормальных формах (4НФ и 5НФ) учитываются не только функциональные, но и многозначные зависимости между полями таблицы. Для их описания познакомимся с понятием полной декомпозиции таблицы.

Полной декомпозицией таблицы называют такую совокупность произвольного числа ее проекций, соединение которых полностью совпадает с содержимым таблицы.

Теперь можно дать определения высших нормальных форм. И сначала будет дано определение для последней из предложенных - 5НФ.

5НФ - Таблица находится в пятой нормальной форме тогда и только тогда, когда в каждой ее полной декомпозиции все проекции содержат возможный ключ. Таблица, не имеющая ни одной полной декомпозиции, также находится в 5НФ.

4НФ - Четвертая нормальная форма является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. Весьма не просто подобрать реальную таблицу, которая находилась бы в 4НФ, но не была бы в 5НФ.

Третья нормальная форма считается оптимальной для небольших баз данных, при проведении дальнейшей нормализации следует учитывать, что при увеличении количества связанных таблиц увеличивается время обработки информации, хранящейся в них. Поэтому разработчик должен сам принимать решение о том, какая ступень нормализации будет оптимальной для проектируемой базы данных.

Вопросы для самоконтроля:

- дайте понятие базы данных;
- что такое реляционная модель данных;
- дайте определение объекта, атрибута, отношения, кортежа, схемы отношений;
- что такое нормализация, нормальных форм;
- 1, 2, 3 нормальные формы. Дайте определение.

ЛАБОРАТОРНАЯ РАБОТА 1.1
РАЗРАБОТКА СТРУКТУРЫ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

Цель: изучение принципов проектирования реляционной БД в соответствии с правилами нормализации.

Материалы и оборудование: ПК

Индивидуальное задание: разработка схемы отношений, соответствующей объекту, описанному в варианте задания, и приведение ее к третьей нормальной форме.

Требования к содержанию отчета

Отчет должен содержать: наименование работы, цель работы, постановку задачи, описание варианта задания, краткие теоретические сведения, описание хода выполнения лабораторной работы, результаты выполнения задания, вывод.

В качестве описания хода выполнения лабораторной работы 1.1 и результата работы должны быть приведены:

- полный список атрибутов разрабатываемой БД с указанием заданных им имен, типа значений, длины, точности;
- описание процесса нормализации: результаты преобразований в нормальные формы с определением первичных и внешних ключей, связей между таблицами, и обоснованием выбранного варианта разделения на таблицы.

Пример выполнения

Пусть поставлена задача: разработать базу данных студентов, хранящую следующую информацию: Ф.И.О. студента, телефон, год поступления, номер зачетки, специальность, код специальности.

Сначала определим набор атрибутов будущей базы. В основном набор и содержание полей будут соответствовать тому списку, который сформулирован в требованиях к базе. Единственная сложность - требование о том, что в базе должны храниться Ф.И.О. студента. Необходимо решить, будут ли они все записываться в одной поле, или фамилия, имя и отчество должных хранится раздельно: т.е. мы должны обеспечить требование атомарности атрибутов базы. Можно предположить, что, как правило, при формировании каких-либо выходных форм и запросов пользователю часто будет удобнее видеть только фамилию и инициалы студента. Если мы будем использовать для ФИО одно поле, то в таком случае потребуется достаточно сложная обработка его содержимого, с выделением в текстовой строке фрагментов, соответствующих составным частями имени. Таким образом, получится, что в одном поле мы пытаемся хранить целую структуру и данный реквизит будет неатомарным. Поэтому следует завести 3 поля: для раздельного хранения фамилии, имени и отчества. Таким образом, набор атрибутов базы будет выглядеть следующим образом (таблица 1.1)

Таблица 1.1 – Атрибуты базы данных «Студенты»

Наименование атрибута	Содержание	Тип	Длина
Fam	Фамилия	Строка	20
Imya	Имя	Строка	15
Otch	Отчество	Строка	25
Tel	Контактный телефон	Строка	13
GodPostup	Год поступления	Дата	-
NomZach	Номер зачетки	Строка	10
Spec	Название специальности	Строка	50
SpecKod	Код специальности	Строка	14

В отношении, состоящем из этих атрибутов, можно выделить первичный ключ – это поле NomZach – номер зачетки.

Таким образом, получившаяся таблица

Students (Fam, Imya, Otch, Tel, GodPostup, NomZach*, Spec, SpecKod)

состоит из атомарных атрибутов и имеет первичный ключ, а следовательно, наша база данных, представленная одной этой таблицей, *находится в 1НФ.*

Для того чтобы таблица соответствовала требованиям 2НФ, все ее неключевые поля должны находиться в полной функционально зависимости от ключа. Иными словами, одному значению первичного ключа должно соответствовать только одно значение неключевого поля.

Это требование выполняется для Fam, Imya, Otch, GodPostup, Spec, SpecKod, но не выполняется для поля Tel, поскольку у одного студента, однозначно определяемого его номером зачетки, может быть несколько контактных телефонов (домашний, мобильный, рабочий телефон родителей и так далее) (рисунок 1.4).

Students

Fam	Imya	Otch	Tel	GodPostup	**NomZach***	Spec	SpecKod

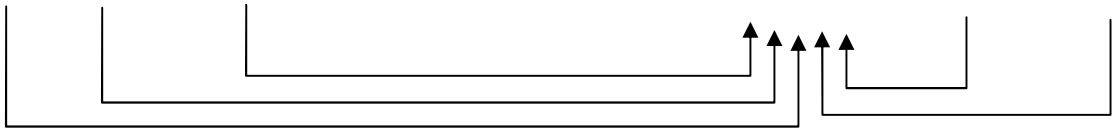


Рисунок 1.4 – Атрибуты таблицы Students, находящиеся в полной функциональной зависимости от первичного ключа

Следовательно, для того, чтобы выполнялись условия соответствия таблицы Students второй нормальной форме, ее нужно разделить на две:

Students (Fam,Imya,Otch,GodPostup, **NomZach*,**Spec,SpecKod)

и

Telefon (**Tel***, Students_NomZach **)

Для таблицы Students первичным ключом будет, естественно, являться поле NomZach.

В таблице Telefon всего два поля: первичным ключом этой таблицы является само поле Tel, а поле Students_NomZach добавлено дополнительно: оно является *внешним ключом* и обеспечивает возможность установить связь данных таблицы Telefon с таблицей Students (рисунок 1.5).

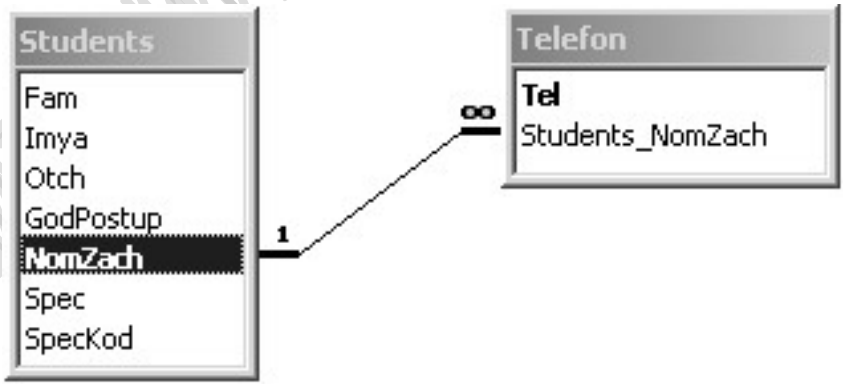


Рисунок 1.5 – Связь между таблицами Telefon и Students

Получившиеся таблицы Telefon и Students *удовлетворяют требованиям 2НФ.*

В таблице Telefon нет транзитивных зависимостей. Следовательно, она удовлетворяет и 3НФ.

Проверим на соответствие требованиям третьей нормальной формы таблицы Students.

В этой таблице есть поля, нарушающие требование 3НФ о том, что ни одно из неключевых полей не должно однозначно идентифицироваться другим неключевым полем: такими полями являются Spec и SpecKod. Действительно, каждая специальность однозначно идентифицируется своим кодом (например, I-53 01 02) и каждому коду соответствует одно официальное наименование специальности (в нашем случае I-53 01 02 «Автоматизированные системы обработки информации»).

В существующем состоянии таблица Students кажется достаточно удобной для хранения данных, однако наличие транзитивной зависимости Spec -> SpecKod приводит к следующим проблемам:

- при добавлении данных о студентах одной и той же специальности в каждой записи будет дублироваться и код, и наименование специальности;
- в случае, если кроме наименования, специальность понадобиться охарактеризовать еще какими-либо атрибутами (например, наименование квалификации, получаемой выпускниками данной специальности), дублирование данных еще больше возрастет;
- при изменении, например, наименования специальности, придется пересмотреть все записи о студентах и исправить значение поля Spec.

Поэтому более разумно и правильно привести эту таблицу также к 3НФ. Для этого поля Spec и SpecKod должны быть выделены в отдельную таблицу. Схема базы данных, получившаяся в результате, приведена на рисунке 1.6

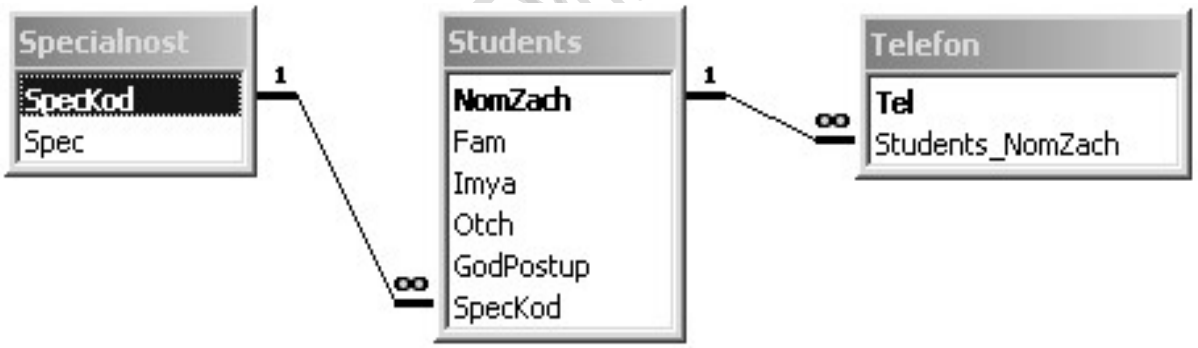


Рисунок 1.6 – Схема базы данных «Студенты»

Все получившиеся таблицы удовлетворяют требованиям 3НФ. Эта степень нормализации вполне достаточна для разрабатываемой базы данных, поэтому будем считать, что процесс проектирования заданной БД завершен.

Контрольные задания по созданию БД

Задание

1. Выбрать вариант задания в соответствии с номером в журнале группы.
2. Уточнив и дополнив заданную предметную область, выявить необходимый набор сущностей, определить требуемый набор атрибутов для каждой сущности, определить связи между объектами.
3. Создать структуры таблиц, ключевые поля. Заполнить таблицы данными. Количество данных в таблицах должно обеспечивать выдачу не менее 3-5 записей по каждому запросу задания. Установить связи между таблицами.
4. Создать формы для ввода информации в удобном для пользователя формате.
5. Создать запросы на выборку в соответствии с заданием. Создать параметрический запрос. Создать запросы на обновление и удаление. Создать перекрестный запрос. Создать запрос для создания отчета.
6. Создать простой отчет и отчет на основе ранее созданного запроса.
7. Создать кнопочную форму для работы со всеми созданными ранее объектами базы данных (таблицы, формы, запросы, отчеты). Предусмотреть в форме выход из базы данных.

Методические указания

Разработанное приложение должно заносить информацию в таблицы созданной базы данных с помощью соответствующих форм, выполнять необходимые действия по модификации и удалению данных в таблицах созданной базы данных с помощью соответствующих форм, поддерживать целостность базы данных, используя соответствующие средства, выполнять запросы из варианта задания и сохранять полученные результаты в соответствующих отчетах.

Предметная область базы данных выбирается в соответствии с вариантом индивидуального задания по номеру в журнале. Для каждого варианта задания приводится минимальный набор характеристик базы данных. Необходимо самостоятельно добавить некоторые характеристики предметной области, позволяющие большим количеством сущностей более полно описать предметную область.

Отчет должен содержать следующее:

- Содержательное описание уточненной предметной области.
- Итерационный процесс построения ER-диаграммы.
- Описание сущностей на языке инфологического проектирования.

- Универсальное отношение, процесс нормализации и реляционная схема, полученная в результате нормализации (3 нормальная форма).
- Следующие распечатки:
  - схема базы данных в форме Access;
  - таблицы, структуры таблиц, формы, отчеты;
  - запросы в режиме Конструктор и на SQL и их результаты;
  - главная кнопочная форма.

Варианты заданий

1. Библиотека

Минимальный список характеристик:

- Автор книги, название, год издания, цена, количество экземпляров, краткая аннотация;
- номер читательского билета, ФИО, адрес и телефон читателя, дата выдачи книги читателю и дата сдачи книги читателем, отметка о выбытии.

Книга имеет много экземпляров и поэтому может быть выдана многим читателям.

Выборки:

- Выбрать книгу, для которой наибольшее количество экземпляров находится "на полках" (не выданы читателям).
- Выбрать читателей, которые имеют задолженность более 4 месяцев.
- Определить книгу, которая была наиболее популярной весной 2000 года.
- Определить читателей, у которых на руках находятся книги на общую сумму более 100 руб.

2. Университет

Минимальный список характеристик:

- Номер, ФИО, адрес и должность преподавателя, ученая степень;
- код, название, количество часов, тип контроля и раздел предмета (дисциплины);
- код, название, номер заведующего кафедрой;
- номер аудитории, где преподаватель читает свой предмет, дата, время, группа.

Один преподаватель может вести несколько дисциплин и одна дисциплина может вестись несколькими преподавателями.

Примечание: Циклы дисциплин: гуманитарный, общеинженерный, математический, компьютерный и т.д.

Выборки:

- Выбрать преподавателя, который был "без работы" весной 2001г.
- Определить возможные "накладки" аудиторий в расписании.
- Вывести расписание занятий группы 'АП-17а' на март 2001г.
- Определить для каждой группы долю дисциплин каждого цикла в процентах.

]

3. Оптовая база

Минимальный список характеристик:

- Код товара, название товара, количество на складе, единица измерения, стоимость единицы товара, примечания - описание товара;
- Номер, адрес, телефон и ФИО поставщика товара, срок поставки и количество товаров в поставке, номер счета.

Один и тот же товар может доставляться несколькими поставщиками и один и тот же поставщик может доставлять несколько видов товаров.
Выборки:

- Выбрать поставщиков, которые поставляют все товары.
- Определить поставщика, который поставляет товар 'коврик для мыши' по самой низкой в среднем цене.
- Вывести названия товаров, цены на которые никогда не повышались.
- Определить, на сколько единиц возросли поставки товара 'инструмент' в 2001г. по сравнению с предыдущим годом.

4. Производство

Минимальный список характеристик:

- Код изделия, название изделия, является ли типовым, примечание - для каких целей предназначено, годовой объем выпуска;
- код, название, адрес и телефон предприятий, выпускающих изделия;
- название, тип, единица измерения материала, цена за единицу, отметка об использовании материала в данном изделии;
- количество материала в спецификации изделия, дата установления спецификации, дата отмены;

- год выпуска и объем выпуска данного изделия предприятием.

Одно изделие может содержать много типов материалов и один и тот же материал может входить в состав разных изделий.

Выборки:

- Определить изделие, в которое входит больше всего материалов типа 'цветной металл'.
- Вывести список изделий, которые не производились в 2000 г.
- Вывести список изделий, для которых затраты на материалы в 2000 г. снизились по сравнению с предыдущим годом.
- Вывести среднемесячный расход материала 'лапша' в 2000 г.

5. Сеть магазинов

Минимальный список характеристик:

- Номер, ФИО, адрес, телефон владельца магазина, размер вклада в магазин, номер регистрации, дата регистрации;
- номер, название, адрес и телефон магазина, уставной капитал, профиль;
- номер, ФИО, адрес, телефон поставщика, а также стоимость поставки данного поставщика в данный магазин.

Один и тот же магазин может иметь несколько владельцев и один и тот же владелец может иметь в собственности много магазинов.
Примечание: профиль - продуктовый, галантерейный, канцелярский и т.п.

Выборки:

- Определить самого молодого предпринимателя, владеющего собственностью в районе 'Киевский'.
- Определить случаи, когда регистрировалось владение лицами, не достигшими 18 лет.
- Определить случаи, когда больше 50% уставного капитала магазина внесено предпринимателем, проживающим в другом районе.
- Вывести список профилей магазинов, которыми владеет предприниматель 'Кузнецов' в порядке убывания вложенного в них капитала

6. Авторемонтные мастерские

Минимальный список характеристик:

- Номер водительских прав, ФИО, адрес и телефон владельца автомобиля;
- номер, ФИО, адрес, телефон и квалификация (разряд) механика;
- номер, марка, мощность, год выпуска и цвет автомобиля;
- номер, название, адрес и телефон ремонтной мастерской;
- стоимость наряда на ремонт, дата выдачи наряда, категория работ, плановая и реальная дата окончания ремонта.

Один и тот же автомобиль может обслуживаться разными автомеханиками и один и тот же автомеханик может обслуживать несколько автомобилей.
Выборки:

- Выбрать фамилию того механика, который чаще всех работает с довоенными автомобилями.
- Выбрать случаи, когда ремонт автомобилей марки 'Мерседес-600' задерживался относительно планового срока.
- Определить тех владельцев автомобилей, которых всегда обслуживает один и тот же механик. Вывести фамилии механика и его постоянного клиента.
- Для каждой категории работ определить, механик какого разряда чаще всего назначается на эту категорию работ

7. Деканат

Минимальный список характеристик:

- Код группы, курс, количество студентов, общий объем часов;
- ФИО преподавателя, вид контроля, дата;
- Название дисциплины, категория, объем часов.

Одна группа изучает несколько дисциплин и одна дисциплина может преподаваться нескольким группам.
Категория дисциплины - гуманитарная, математическая, компьютерная, общеинженерная и т.д.
Вид контроля - зачет, экзамен.

Выборки:

- Для каждой группы определить продолжительность сессии.
- Определить преподавателя, который в сессию принимает экзамены (не зачеты) у наибольшего числа студентов.
- Определить, какой процент от общего объема дисциплин, изучаемых группой 'АП-17а', составляют дисциплины каждой категории.
- Определить, не сдает ли какая-либо группа два экзамена в один день.

8. Договорная деятельность организации

Минимальный список характеристик:

- Шифр работы, название, трудоемкость, дата завершения;
- ФИО сотрудника, должность, табельный номер;
- Дата выдачи поручения на работу, трудоемкость, плановая и реальная даты окончания.

Одна и та же работа может выполняться несколькими сотрудниками и один и тот же сотрудник может участвовать в нескольких работах.
Выборки:

- Определить те работы в 2000 г., по которым плановые сроки выполнения превышают заданную дату завершения.
- Определить общее количество работ, находившихся на выполнении у некоторого сотрудника весной 1999 г.
- Определить те работы, которые к дате завершения были выполнены не более, чем на 50%.
- Определить должностной состав сотрудников, выполняющих работу 'проект Гелиограф'.

9. Поликлиника

Минимальный список характеристик:

- Номер, фамилия, имя, отчество, дата рождения пациента, социальный статус, текущее состояние;
- ФИО, должность, квалификация и специализация лечащего врача;
- диагноз, поставленный данным врачом данному пациенту, необходимо ли амбулаторное лечение, срок потери трудоспособности, состоит ли на диспансерном учете, дата начала лечения.

Текущее состояние - лечится, вылечился, направлен в стационар, умер.

Социальный статус пациента - учащийся, работающий, временно неработающий, инвалид, пенсионер

Специализация врача - терапевт, невропатолог и т.п.

Квалификация врача - 1-я, 2-я, 3-я категория.

Один и тот же пациент может лечиться у нескольких врачей и один врач может лечить несколько пациентов.
Выборки:

- Определить те случаи, когда заболевание 'язва желудка' лечилось врачом специализации 'невропатолог'.
- Вывести имена тех врачей, которые работают исключительно с пенсионерами.

- Определить процент смертности от заболевания 'кариес'.
- Пациентов, которые болеют (болели) всеми болезнями.

10. Телефонная станция

Минимальный список характеристик:

- Номер абонента, фамилия абонента, адрес, наличие блокиратора, примечание;
- Код АТС, код района, количество номеров;
- Номер спаренного телефона абонента, задолженность, дата установки.

Один спаренный номер одной АТС может использоваться несколькими абонентами и один и тот же абонент может использовать телефоны разных АТС.
Выборки:

- Выбрать пары сблокированных телефонов.
- Определить АТС, районы действия которых перекрываются.
- Выбрать телефоны группового пользования, Вывести их номера и фамилии абонентов.
- Выбрать список абонентов АТС 47, имеющих задолженность больше 100 руб.

11. Спорт

Минимальный список характеристик:

- Название вида спорта, единица измерения, мировой рекорд и его дата;
- ФИО спортсмена, год рождения, команд, спортивный разряд;
- Наименование соревнования, показанный результат, дата проведения, место проведения.

Один спортсмен может заниматься разными видами спорта.
Выборки:

- Вывести таблицу распределения мест в соревновании 'открытый чемпионат' в городе 'Киев' по 'шахматам' в 2000 г.
- Определить спортсменов, которые выступают более чем в 3 видах спорта.
- Вывести список спортсменов, превысивших мировые рекорды.
- Определить наилучший показатель спортсмена 'Караваев' в виде спорта 'бег'.

12. Сельскохозяйственные работы

Минимальный список характеристик:

- Наименование сельскохозяйственного предприятия, дата регистрации, вид собственности, число работников, основной вид продукции, является ли передовым в освоении новой технологии, прибыль, примечание;
- Дата поставки, объем, себестоимость поставщика;

- Название продукции, единица измерения, закупочная цена.

Одно и то же предприятие может выпускать разные виды продукции и одна и та же продукция может выпускаться разными предприятиями.
Выборки:

- Вывести предприятия, являющиеся нерентабельными в текущем году.
- Определить, какого вида собственности предприятие является ведущим в поставке продукта 'банан'.
- Определить объем дохода на одного работника в предприятии 'КАМАЗ'.
- Вывести список продуктов, для которых закупочная цена, как правило, ниже себестоимости производителя.

13. Городской транспорт

Минимальный список характеристик:

- Вид транспорта, средняя скорость движения, количество машин в парке, стоимость проезда;
- номер маршрута, количество остановок в пути, количество машин на маршруте, количество пассажиров в день;
- начальный пункт пути, конечный пункт, расстояние.

Один и тот же вид транспорта может на разных маршрутах использовать разные пути следования.
Выборки:

- Определить оптимальный по времени маршрут между пунктами 'Холодная Гора' и 'Парк '.
- Определить среднее время ожидания на остановке троллейбуса №39.
- Вывести маршруты трамваев в порядке убывания их протяженности.
- Вывести список ежедневных денежных поступлений для всех видов транспорта.

14. География

Минимальный список характеристик:

- Название страны, регион, столица, площадь территории, является ли страна развитой в экономическом отношении;
- количество населения,
- название национальности, язык, общая численность.

В одной стране могут присутствовать люди разной национальности.
Выборки:

- Считая, что государственным является язык, на котором разговаривает не менее 20% населения страны, выбрать список государственных языков страны 'Китай'.
- Выбрать численность населения по всем странам.
- Определить столицу той страны, где проживает более всего представителей национальности 'светлый эльф'.
- Выбрать список национальностей, проживающих в регионе 'Драконовы Горы'

15. Домоуправление

Минимальный список характеристик:

- Номер квартиры, номер дома, число жильцов, площадь;
- Вид оплаты, цена за единицу площади, цена за одного жильца;
- Сумма оплаты, месяц и год оплаты, дата оплаты.

  В одной квартире используются разные виды оплаты.

Выборки:

- Определить сумму месячной оплаты для всех квартир дома №5.
- Определить задолженность по оплате 'теплоснабжения' квартиры №512 дома №5.
- Определить общее число жильцов дома №5.
- Выбрать список квартир, которые не имеют задолженностей на начало текущего года.

16. Аэропорт

Минимальный список характеристик:

- Номер самолета, тип, число мест, скорость полета;
- Номер маршрута, расстояние, пункт вылета, пункт назначения;
- Дата и время вылета, дата и время прилета, количество проданных билетов.

Один тип самолета может летать на разных маршрутах и по одному маршруту могут летать разные типы самолетов.

Выборки:

- Определить среднее расчетное время полета для самолета 'ТУ-154' по маршруту 'Чугуев' - 'Мерефа'.
- Выбрать марку самолета, которая чаще всего летает по тому же маршруту.
- Выбрать маршрут/маршруты, по которым чаще всего летают рейсы, заполненные менее чем на 70%.
- Определить наличие свободных мест на рейс №870 31 декабря 2000 г.

17. Персональные ЭВМ

Минимальный список характеристик:

- Фирма-изготовитель, название и место размещения фирмы;
- Тип процессора, тактовая частота, объем ОЗУ, объем жесткого диска, дата выпуска ПЭВМ;
- Фирма-реализатор: наименование, адрес, телефон;
- Объем партии рыночного предложения, цена партии.

Один тип персональной ЭВМ (ПЭВМ) может изготавливаться разными фирмами и одна и та же фирма может собирать разные типы ПЭВМ.

Выборки:

- Определить фирму, которая представляет самую новую модель на базе процессора 'Pentium-IV'.
- Выбрать модель с наибольшей тактовой частотой, которая выпускается в г.'Богодухов'.
- Определить фирму, которая представляет на рынки товары на наибольшую сумму.
- Выбрать города, в которых выпускаются ПЭВМ на базе процессора 'POWER-4'.

18. Личные данные о студентах.

Минимальный список характеристик:

- ФИО студента, курс, факультет, специальность, дата рождения студента, семейное положение, сведения о семье;
- Название факультета, число мест на факультете;
- Номер группы, сумма стипендии студента, год зачисления.

Один и тот же студент может обучаться на разных факультетах.

Выборки:

- Выбрать количество студентов на каждом курсе

- Выбрать суммарную стипендию группы '109' факультета 'АП'.
- Выбрать список групп факультета 'АП' с указанием численности студентов в каждой группе.
- Выбрать список студентов, не достигших к моменту зачисления 18 лет.

19. Зоопарк

Минимальный список характеристик:

- Название вида животного, суточное потребление корма, семейство, континент обитания.
- Название комплекса, номер помещения, наличие водоема, отопления, количество животных в помещении.

Один и тот же вид животного может в зоопарке находиться в разных помещениях и в одном помещении может находиться несколько видов животных.

Выборки:

- Определить суточное потребление корма обитателями комплекса 'приматы'.
- Выбрать случаи размножения животного вида 'карликовый гиппопотам' в помещении без водоема.
- Определить общую численность представителей семейства 'псовые' в зоопарке.
- Вывести все пары видов, которые содержатся в одном помещении

20. Шахматы

Минимальный список характеристик:

- Фамилия спортсмена, дата рождения, страна, спортивный разряд, участвовал ли в борьбе за звание чемпиона мира, рейтинг, примечание
- Турнир, страна, город, дата проведения, уровень турнира;
- Стартовый номер спортсмена в данном турнире, занятое место.

Один шахматист может участвовать в разных турнирах.

Выборки:

- Выбрать турнир с самым высоким рейтингом участников.
- Выбрать те турниры, где все призовые места заняли представители страны-хозяина турнира.
- Выбрать тех шахматистов, которые заняли не менее трех призовых мест в течение 2000 г.
- Определить турниры, в которых участник с самым высоким рейтингом занял последнее место.

21. Судоходство.

Минимальный список характеристик:

- Название корабля, водоизмещение, порт приписки, капитан и т.д.

- Название порта, страна, категория;
- Дата посещения порта, дата убытия, номер причала, цель посещения.

Один корабль может посещать несколько портов.

Выборки:

- Выбрать список кораблей, посещавших 'Одессу' зимой 1998/99 г.
- Определить, когда корабль 'Кузнецов' посещал порт 'Новороссийск' с целью 'починки такелажа'.
- Определить страны, в которые никогда не приходят корабли с целью 'туризм'.
- Определить, с какой целью чаще всего заходят корабли в порт 'Ялта'.

22. Научные конференции.

Минимальный список характеристик:

- Имя ученого, организация, страна, ученая степень;
- Название конференций, место проведения, дата;
- Тип участия, тема доклада, публикация (да/нет).

Один ученый может участвовать в разных конференциях и с разными типами докладов. Тип участия: доклад, сообщение, стендовый доклад, оргкомитет и т.д.

Выборки:

- Выбрать список ученых, имевших публикации в 1999 г. с указанием числа публикаций для каждого.
- Выбрать названия конференций, материалы которых не опубликованы.
- Определить, в какой конференции участвовало больше всего докторов наук.
- Выбрать список конференций с указанием числа представленных на каждой из них стран.

23. Программные продукты.

Минимальный список характеристик:

- Название продукта, версия, тип, фирма, дата выпуска, прикладная область, стоимость лицензии;
- Название пользователя, регион, сфера применения;
- Стоимость инсталляции, дата инсталляции, дата деинсталляции, количество лицензий при инсталляции.

Один и тот же программный продукт может инсталлироваться (деинсталлироваться) разными пользователями и один пользователь может инсталлировать (деинсталлировать) разные продукты.

Прикладная область: делопроизводство, управление технологическим процессом, е-коммерция, е-бизнес и т.д.

Тип программного продукта: ОС, сервер приложений, СУБД, Web-сервер, система программирования и т.д.

Выборки:

- Определить прикладную область, которая требует наибольшей номенклатуры программных продуктов.
- Определить затраты на приобретение/модификацию программного обеспечения, сделанные покупателем 'Белый ветер' в 2000г.
- Выбрать список продуктов типа 'серверные операционные системы', в порядке убывания их популярности.

24. Операционная система

Минимальный список характеристик:

- Название процесса, приоритет, класс, идентификатор владельца;
- Название ресурса, количество, цена за единицу;
- Запланированный ресурс, количество, запрошено/выделено.

Один и тот же процесс может задействовать много разных ресурсов и один и тот же ресурс может быть задействован разными процессами.

Примечание:

Классы процессов (в порядке убывания абсолютного приоритета): критический, серверный, нормальный, запасной.

Выборки:

- Определить, есть ли в системе процессы с запросами, превышающими возможности системы.
- Выбрать очередь к ресурсу 'файл data1' в порядке убывания приоритетов.
- Определить, в очередях к каким ресурсам есть процессы с приоритетом выше, чем у тех, которые владеют ресурсами.
- Определить владельца, у которого "самые большие аппетиты" в ценовом выражении.

25. Добыча полезных ископаемых

Минимальный список характеристик:

- Полезное ископаемое, единица измерения, годовая потребность, цена за единицу, тип;

- Название месторождения, запасы, способ разработки, годовая добыча, себестоимость за единицу;
- пункт вывоза, пропускная способность путей сообщения данного пункта.

Из одного и того же пункта вывоза могут вывозиться разные полезные ископаемые и одно и то же полезное ископаемое может вывозиться с разных пунктов вывоза.
Выборки:

- Считая, что показатели даны на текущий год, а ежегодная добыча будет возрастать на 10% каждый год, определить те месторождения, которые будут исчерпаны через 5 лет.
- Определить те ископаемые, потребность в которых не удовлетворяется.
- Определить, какие ископаемые добываются в 'Эльдорадо', и добыча каких из них является прибыльной.
- Выбрать список мест, в которых добывается 'твердое топливо' 'открытым способом'.

26. Автотранспортное предприятие

Минимальный список характеристик:

- номерной знак автомобиля, марка автомобиля, его техническое состояние, пробег, грузоподъемность, расход топлива,
- табельный номер водителя, ФИО, дата рождения, стаж работы, оклад, категория;
- дата выезда, дата прибытия, место назначения, расстояние, расход горючего, масса груза.

Один и тот же автомобиль может использоваться разными водителями и один водитель может использовать разные автомобили.
Выборки:

- Выбрать автомобиль с наименьшим расходом горючего за данный период.
- Выбрать водителей, использующих заданную марку автомобиля.
- Подсчитать количество автомобилей, имеющих плохое техническое состояние.
- Выбрать водителей, которые чаще всего ездят по данному маршруту.

27. Театр

Минимальный список характеристик:

- Актер, ФИО, звание, амплуа, пол;
- Дата назначения на роль, дата снятия с роли, тип роли, режиссер, номер состава;
- Название роли, тип (амплуа) роли, название пьесы.

  На одну и ту же роль могут назначаться разные актеры.

Амплуа: герой-любовник, инженю, злодей т.д.

Тип роли: главная, вторая, эпизод, статист т.д.

Выборки:

- Определить любимого актера режиссера Балаяна.
- Выбрать имена актеров, в творческой биографии которых более 50% ролей назывались 'кушать подано'.
- Выбрать список пьес, в которых исполнители главных ролей менялись более 3 раз.
- Выбрать список актеров, которые находятся в "творческом простое" с начала 2000 г.

28. Справочная аптек

Минимальный список характеристик:

- Название лекарства, показания к использованию, противопоказания, производитель;
- Наличие лекарства, тип, дозировка, цена, количество, срок годности;
- Номер аптеки, специализация аптеки, район, телефон, и т.п.

Тип: таблетки, микстура, мазь и т.д.

Выборки:

- По ассортименту предлагаемых лекарств определить, какой болезнью чаще всего страдают жители района 'Киевский'.
- Определить, какие убытки понесет аптека №47, если в течение месяца не реализует все лекарства, у которых кончается срок годности.
- Определить, в каких аптеках дешевле всего 'анальгин'.
- Выбрать список лекарств, которые подходят для больного, страдающего болезнями 'цирроз печени' и 'ветрянка' одновременно.

29. Кулинария

Минимальный список характеристик:

- Название блюда, категория, рецепт, вес порции;
- Название продукта, категория, калорийность, цена за ед., ед. измерения.
- Состав блюда, количество, очередность добавления, на сколько порций.

Одно блюдо состоит из разных продуктов и один и тот же продукт может входить в состав разных блюд.

Категория блюда: первое, второе, гарнир, десерт и т.д.

Категория продукта: мучное изделие, мясо, молоко, фрукты и т.д.

Выборки:

- Определить, для каких блюд продукты категории 'овощи' предварительно подвергаются 'пассировке'.
- Выбрать названия блюд с указанием калорийности одной порции для каждого из них.
- Определить блюдо, в которое входит больше всего продуктов категории 'пряность'.
- Для всех блюд категории 'первое блюдо' выбрать списки входящих в них продуктов в порядке их добавления.

30. Изучение студентами дисциплин по выбору.

Минимальный список характеристик:

- Фамилия студента, адрес, телефон, номер зачетной книжки
- Деканат, адрес деканата.
- Номер группы, специальность в группе.
- Наименование дисциплины, количество лекционных часов, семинарских и лабораторных занятий.
- Отметка о сдаче дисциплины.

Каждый студент должен изучить несколько дисциплин по выбору и каждая дисциплина может изучаться несколькими студентами.

Выборки:

- Получить список студентов, изучающих заданную дисциплину и сдавших ее.
- Получить список дисциплин, изученных заданным студентом и объем дисциплины.
- Получить список адресов деканатов.

ТЕМА 2 СУБД MICROSOFT ACCESS 2000

Основные понятия по теме

1 Основные характеристики Access 2000.

2 Интерфейс Access.
3 Объекты Access. Диалоговые средства конструирования объектов.

4 Создание файла базы данных Access. Создание таблицы базы данных.

5. Имена полей и типы данных.

6. Разработка форм, отчетов и макросов в Access 2000.

Программа Microsoft Access 2000 входит в состав программного пакета Microsoft Office 2000 и представляет собой мощную систему, обеспечивающую эффективную разработку и сопровождение баз данных.

СУБД Access ориентирована на работу с объектами, к которым относятся таблицы базы данных, запросы, а также объекты приложений для работы с базой данных: формы, отчеты, страницы, макросы и модули.

Множество мастеров Access позволяют автоматизировать процесс создания таблиц базы данных, форм, запросов, отчетов и страниц доступа к данным; анализировать таблицы БД и выполнять многие другие работы. Практически для любых работ имеется мастер (wizard),который поможет их выполнить.

В СУБД Access процесс создания реляционной базы данных включает создание схемы данных. Схема данных наглядно отображает таблицы и связи между ними, а также обеспечивает использование связей при обработке данных.

Access может использовать данные различных СУБД. Непосредственно могут обрабатываться файлы Paradox, dBase, FoxPro, а также базы данных, поддерживающие стандарт открытого доступа к данным (Open Database Connectivity, ODBC): Oracle, Microsoft SQL Server, DB2, Sybase SQL Server и др.

СУБД Microsoft Access является системой управления реляционной базой данных, включающей все необходимые инструментальные средства для создания локальной базы данных, общей базы данных в локальной сети с файловым сервером или создания приложения пользователя, работающего с базой данных на SQL - сервере.

Диспетчером данных, выполняющим загрузку и сохранение данных в базе данных пользователя и системных базах данных, является ядро базы данных Microsoft Jet. Access 2000 построена на основе усовершенствованной версии ядра базы данных Microsoft Jet 4.0. Эта версия имеет более высокую производительность и улучшенные сетевые характеристики.

Access 2000 входит в состав Microsoft Office 2000 (в варианты Professional, Premium и Developer) и, как и другие компоненты Office 2000,аботает в среде Windows 95, Windows 98 или Windows NT Workstation 4.0 и выше.

Практическим минимумом, предъявляемым Access 2000 к персональному компьютеру, является Pentium 75 MHz и 16 Мб оперативной памяти при работе под Windows 95 или Windows 98 или 32 Мб при работе под Windows NT Workstation.

В Access обеспечивается все возможности динамического обмена данными (Dynamic Data Exchange, DDE) с любым приложением Windows, поддерживающим DDE. Access поддерживает также механизм связывания и внедрения объектов (Object Linking and Embedding, OLE),

обеспечивающий установление связи с объектами другого приложения или внедрение объекта в базу данных. Активизация внедренного объекта открывает программу, которая его создала, и пользователь может изменить объект. При установлении связи с объектом он по-прежнему сохраняется в файле объекта, а не в базе данных. За счет этого он может обновляться независимо, а в базе данных всегда будет представлена последняя версия объекта.

Внедряемыми или связываемыми объектами могут быть документы различных приложений Windows: рисунки, графики, электронные таблицы или звуковой файл.

Access может использовать данные различных СУБД. Непосредственно могут обрабатываться файлы Paradox, dBase, FoxPro, а также базы данных, поддерживающие стандарт открытого доступа к данным (Open Database Connectivity, ODBC): Oracle, Microsoft SQL Server, DB2, Sybase SQL Server и др.

Access - это типичная настольная база данных. В то же время на небольшом предприятии с количеством компьютеров не больше 10, ресурсов Access вполне может хватить для обслуживания всего делопроизводства, естественно, в связке с Microsoft Office. То есть все пользователи могут обращаться к одной базе данных, установленной на одной рабочей станции, которая не обязательно должна быть выделенным сервером. Для того чтобы не возникали проблемы сохранности и доступа к данным, имеет смысл воспользоваться средствами защиты, которые предоставляет Access. При этом вы можете воспользоваться Мастером, если не уверены, что сами правильно установите права и ограничения для пользователей.

В отличие от большинства средств разработки, СУБД Access имеет русифицированный интерфейс и частично переведенный на русский язык файл контекстной помощи.

Access имеет характерный для всех приложений Microsoft Windows удобный графический интерфейс, ориентированный на комфортную работу пользователя. Для работы с таблицами базы данных и другими объектами Access предоставляет многочисленные команды меню и контекстно-зависимые панели инструментов. Поскольку интерфейс приложений Microsoft Office унифицирован, пользователю требуется меньше времени на освоение приложения.

Пользователь имеет возможность с помощью мыши можно переносить объекты между различными базами данных. При этом необходимо запустить две задачи Microsoft Access. Возможен перенос таблиц и запросов Access в другие приложения, например, в Microsoft Word и Microsoft Excel.

СУБД Access ориентирована на работу с объектами, к которым относятся таблицы базы данных, запросы, а также объекты приложений для работы с базой данных: формы, отчеты, страницы, макросы и модули.

Для типовых процессов обработки данных - просмотра, обновления, поиска по заданным критериям, получения отчетов - в Access имеются средства конструирования форм, запросов, отчетов и страниц. Объекты приложений состоят из графических элементов, называемых элементами управления. Основные элементы управления служат для связи объектов с записями таблиц, являющихся источниками данных.

При создании приложений пользователя также используются средства программирования, реализуемые объектами другого типа - макросами и модулями на языке программирования Visual Basic for Applications (VBA).

Каждый объект и элемент управления имеет свои свойства, определяя которые, можно настраивать объекты и элементы управления. С каждым объектом и элементом управления связывается набор событий, которые могут обрабатываться макросами или процедурами на VBA.

Объекты представлены в окне базы данных Access. Все операции по работе с объектами базы данных и приложений начинаются в этом окне.

Рисунок 3.1 – Основное окно конструирования БД

Таблицы (Tables) создаются пользователем для хранения данных об одном информационном объекте модели данных предметной области. Таблица состоит из полей (столбцов) и записей (строк). Каждое поле содержит одну характеристику объекта предметной области. В записи собраны сведения об одном экземпляре этого объекта.

Запросы (Queries) создаются пользователем для выборки нужных данных из одной или нескольких связанных таблиц. Результатом выполнения запроса является таблица, которая может быть использована наряду с другими таблицами БД при обработке данных. Запрос может формироваться в виде запросов по образцу (QBE) или с помощью инструкции SQL - языка структурированных запросов. С помощью запроса можно также обновить, удалить или добавить данные в таблицы или создать новые таблицы на основе уже существующих.

Формы (Forms) являются основным средством создания диалогового интерфейса приложения пользователя. Форма может создаваться для ввода и просмотра взаимосвязанных данных базы на экране в удобном виде, который соответствует привычному для пользователя документу. Формы также могут использоваться для создания панелей управления в приложении.

Отчеты (Reports) предназначены для формирования выходных документов, содержащих результаты решения задач пользователя, и вывода их на печать.

Страницы (Pages) - Страницы доступа к данным являются диалоговыми Web-страницами, которые поддерживают динамическую связь с базой данных и позволяют просматривать, редактировать и вводить данные в базу, работая в окне браузера.

Макросы (Macros). Макрос является программой, которая содержит описание последовательности действий, выполняемых при наступлении некоторого события в объекте или элементе управления приложения. Каждое действие реализуется макрокомандой. Создание макросов осуществляется в диалоговом режиме путем выбора нужных макрокоманд и задания параметров, используемых ими при выполнении.

Модули (Modules) содержат процедуры на языке VBA. Могут создаваться процедуры-функции, которые разрабатываются пользователем для реализации нестандартных функций в приложении пользователя, и процедуры для обработки событий. В Access 2000 для удобства пользователя объекты базы данных могут быть объединены в группы по функциональному или иному признаку. Группы содержат ссылки на объекты базы данных различных типов.

В окне базы данных Access 2000 наряду со списком созданных объектов представлены ярлыки (shortcuts), которые предназначены для быстрого запуска мастеров или конструктора создания нового объекта.

Все таблицы базы данных, а также другие объекты Access: формы, запросы, отчеты, макросы и модули, построенные для этой базы, и внедренные объекты могут размещаться на диске в одном

файле формата .mdb. Это упрощает технологию ведения базы данных и приложения пользователя. Обеспечивается высокая компактность размещения всех объектов БД на диске и эффективность обработки данных. Страницы доступа к данным Access сохраняются в отдельных файлах, в файле БД размещаются только ссылки на них.

Access предоставляет в распоряжение непрограммирующего пользователя разнообразные диалоговые средства, которые позволяют ему создавать приложения, не прибегая к разработке запросов на языке SQL или к программированию макросов или модулей на языке VBA.

Для автоматизации создания объектов БД - таблиц, запросов по примеру (Query By Example, QBE), схемы базы данных, и объектов приложения (форм, отчетов, страниц) используются специализированные диалоговые средства, называемые конструктором (Design). Конструктор предоставляет пользователю набор инструментов, с помощью которых можно быстро создать и модифицировать объект. Для конструирования макета форм, отчетов и страниц используется панель элементов, которая появляется при вызове конструктора.

Предусмотрено автоматическое конструирование форм, запросов, отчетов, страниц и их элементов с помощью программ-мастеров и команд, начинающихся с приставки "авто".

В Access 2000 для упрощения внесения изменений в объекты базы данных; разработана технология интеллектуальной замены имен объектов в базе данных. При этом автоматически исправляются ошибки, вызванные переименованием таблиц, полей, форм, отчетов, запросов, текстовых блоков или других элементов управления. Реализуется за счет того, что каждый именуемый объект (или элемент) базы данных имеет внутренний уникальный идентификатор, имя является только псевдонимом. При переименованиях изменяется лишь псевдоним и при необходимости корректируются все ссылки на объект из других объектов. Для применения этой технологии следует до создания объектов установить соответствующие параметры в разделе Автозамена имен (Name AutoCorrect) на закладке Общие (General), открываемой через меню Сервис|Параметры (Tools|Options).

На компьютере пользователя, который будет работать с СУБД Access, должна быть установлена операционная система Windows 95/98/NT и СУБД Access. Для того чтобы начать работу в СУБД Access, можно, например, после загрузки операционной системы в нижней части рабочего стола на Панели задач нажать кнопку Пуск и , открыв в главном меню Windows пункт Программы, выбрать программу Microsoft Access и запустить ее.

После запуска Microsoft Access одновременно с его окном выводится первое диалоговое окно, позволяющее начать создание новой базы данных или открыть существующую.

Диалоговое окно появляется, если в окне Параметры (Options), вызываемом по команде меню Сервис |Параметры (Tools|Options), на вкладке Вид (View) в группе Отображать установлен флажок Окно запуска (Startup Dialog Box).

Существующую базу данных можно открыть, выбрав ее из списка в диалоговом окне. Если это окно не появляется при запуске Access, для открытия БД выполняется команда Файл|Открыть (File|Open) или нажимается кнопка Открыть (Open). После выполнения этой команды открывается окно базы данных.

Рисунок 3.2 – Диалог создания новой БД

В окне базы данных представлены два раздела: Раздел Объекты (Objects) с основными типами объектов базы данных: Таблицы (Tables), Запросы (Queries), Формы (Forms), Отчеты (Reports), Страницы (Pages), Макросы (Macros), Модули (Modules) и раздел Группы (Groups), где создаются пользовательские группы объектов, предназначенные для хранения ссылок на объекты различных типов, объединенные, например, по функциональному назначению

Рабочее поле окна базы данных предназначено для отображения списка объектов выбранного типа. Кроме того, В Access 2000 здесь размещены ярлыки, открывающие возможность сразу приступить к созданию объекта в основных режимах: в режиме конструирования и с помощью мастера.

При открытии окна БД по умолчанию выводится панель инструментов База данных (Database). Эта же панель инструментов выводится в окне Access до открытия базы данных, но большинство ее кнопок являются недоступными.

Диалоговое окно позволяет начать создание новой базы данных выбором параметра Новая база данных (New Database). Если первое диалоговое окно не выводится, можно начать создание базы данных с помощью команды Файл|Создать (File|New) или кнопки Создать (New) на панели инструментов База данных (Database). Независимо от выбранного варианта начала создания базы данных, Access выведет окно Создание (New).

Рисунок 3.3 – Окно создания базы данных

Пользователь выбирает вкладку Базы данных, чтобы создать одну из типовых баз данных с помощью мастера на основе существующих шаблонов. На закладке представлены шаблоны таких баз данных. Чтобы начать работу мастера, достаточно щелкнуть на значке нужной базы данных.

Вкладка Общие (General) содержит значок База данных (Blank Database), позволяющий приступить к созданию новой оригинальной базы данных. Access 2000 дополнен новыми

средствами, предназначенными для создания проекта приложения, работающего с базой данных SQL-сервера, и страниц доступа к данным Access. На вкладке Общие (General) имеются соответствующие значки: Проект (новая база данных) - Project (New Database), Проект (существующая база данных) - Project (Existing Database), Страница доступа к данным (Data Access Page).

Проект (существующая база данных) и Проект (новая база данных) позволяют создать проект - приложение пользователя, которое работает с базой данных, размещенной на SQL-сервере.

Страница доступа к данным (Data Access Page) позволяет создавать Web-страницы специального типа, предназначенные для просмотра и работы с данными в базах Microsoft Access или Microsoft SQL Server из Internet.

Access хранит все таблицы базы данных, а также другие объекты в одном файле. Прежде чем приступить к созданию таблиц БД, необходимо создать файл пустой базы данных.

Для создания файла новой БД надо в окне Создание (New) выбрать вкладку Общие (General) дважды щелкнуть на значке База данных (Blank Database). В раскрывающемся списке Папка (Save in) появившегося окна Файл новой базы данных (File New Database) нужно выбрать каталог, в котором будет размещен файл, задать имя файла новой базы данных и нажать кнопку Создать (Create). В результате открывается окно новой базы данных <имя БД> : база данных (Database).

Задавая имя файла базы данных, следует иметь ввиду, что его предельная длина составляет 255 символов, включая пробелы. Имена файлов не должны содержать следующих символов: \ ? : * ? " < > | .

Тип файла по умолчанию имеет значение Базы данных Microsoft Access (расширение *.mdb, Microsoft Access Databases), что приводит к созданию файла базы данных, имеющего расширение .mdb. Это расширение является зарегистрированным в Windows для данного типа файлов и связывается с программой Access.

В результате выполнения команды Создать (New) открывается окно новой базы данных <имя БД> : база данных (Database). Причем <имя БД> соответствует заданному в окне Файл новой базы данных (File New Database).

В окне новой базы данных в разделе Объекты (Objects) вертикальным рядом кнопок представлены все объекты, которые могут быть созданы в БД: таблицы, запросы, отчеты, страницы, макросы и модули. При нажатии кнопки в рабочем поле окна отображается список имен объектов данного типа. При создании новой базы данных список для любого выбранного типа объекта отсутствует.

Объекты различных типов могут объединяться в группы, которые представлены в разделе Группы (Groups). Группы позволяют в больших БД объединить объекты одной темы. Изначально в разделе Группы (Groups) существует единственная группа Избранное (Favorites). Для создания новой группы необходимо нажать правую кнопку на строке этой группы и выбрать из контекстного меню команду Новая группа (New Groups). Для внесения объектов в группу выделяется нужный объект, вызывается контекстное меню, выбирается команда Добавить в группу (Add to Group) и в ней группа, в которую включается объект. Объекты представляются в группе ярлыками, ссылающимися на включенный в группу объект. При выполнении этой команды также можно создать новую группу

Создание таблицы БД состоит из двух этапов. На первом этапе определяется ее структура: состав полей, их имена, последовательность размещения полей в таблице, тип данных каждого поля, размер поля, ключи, индексы таблицы и другие свойства полей. На втором этапе производится создание записей таблицы и заполнение их данными.

Для создания новой таблицы надо в окне базы данных выбрать объект Таблицы (Tables) и нажать кнопку Создать (New). В открывшемся окне Новая таблица (New Table) нужно выбрать один из режимов создания таблицы (рис). В Access 2000 основные первые три режима вынесены в рабочее поле, предназначенное для отображения списка таблиц. Это позволяет

сразу перейти в нужный режим создания таблицы, сократив число выполняемых пользователем операций.

Строка Создание таблицы в режиме конструктора (Create table Design View) в рабочем поле окна базы данных или Конструктор (Design View) в окне Новая таблица (New Table) определяет выбор основного способа создания новой таблицы, при котором создание таблицы начинается с определения ее структуры в режиме конструктора таблиц. В режиме конструктора пользователь может сам установить параметры всех элементов структуры таблицы.

При переходе в режим конструктора таблиц меняется состав команд меню, а панель инструментов базы данных заменяется на панель инструментов Конструктор таблиц (Table Design).

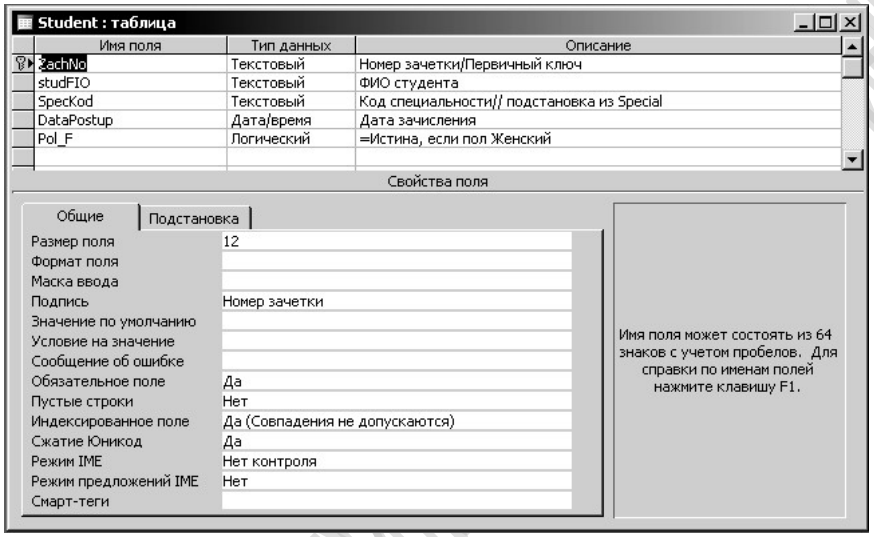


Рисунок 3.4 – Диалоговое окно Конструктора таблиц

Для определения поля в окне Таблица (Table) задаются Имя поля (Field Name), Тип данных (Data Type), Описание (Description) - краткий коммента-рий, а также свойства поля в разделе Свойства поля (Field Properties). На вкладке Общие (General) представлены строки свойств поля, в том числе максимальный размер, подпись, которая выводится в заголовке столбца, значение по умолчанию и др. На вкладке Подстановка (Lookup) выбирается тип элемента управления (Display Control): поле, список, или поле со списком.

Имя поля (Field Name). Каждое поле в таблице должно иметь уникальное имя, удовлетворяющее соглашениям об именах объектов в Access. Оно является комбинацией из букв, цифр, пробелов и специальных символов, за исключением символов " . " " ! " " ' " " [" "] " . Имя не может начинаться с пробела и содержать управляющие символы с кодами ASCII от 00 до 31. Максимальная длина имени 64 символа.

Тип данных (Data Type). Тип данных определяется значениями, которые предполагается вводить в поле, и операциями, которые будут выполняется с этими значениями. В Access допускается использование девяти типов данных. Список возможных типов данных вызывается нажатием кнопки списка при выборе типа данных каждого поля:

Текстовый (Text) - тип данных по умолчанию. Текст или цифры, не участвующие в расчетах. Число символов в поле не должно превышать 255. Максимальное число символов, которое можно ввести в поле, задается в свойстве Размер поля (FieldSize). Пустые символы в неиспользуемой части поля не сохраняются

Поле МЕМО (Memo). Длительный текст, например, некоторое описание или примечание. Максимальная длина 64 000 символов

Числовой (Number). Числовые данные, используемые в математических вычислениях. Конкретные варианты числового типа и их длина задаются в свойстве Размер поля (FieldSize). Для проведения денежных расчетов определен другой тип данных - Денежный (Currency)

Денежный (Currency). Денежные значения и числовые данные, используемые в расчетах, проводящихся с точностью до 15 знаков в целой и до 4 знаков в дробной части. Длина поля 8 байт. При обработке числовых значений из денежных полей выполняются вычисления с фиксированной точкой более быстрые, чем вычисления для полей с плавающей точкой, кроме того, при вычислениях предотвращается округление. Учитывая эти обстоятельства, рекомендуется для полей, в которых планируется хранить числовые значения с указанной точностью, использовать денежный тип данных

Дата/время (Data/Time). Значения даты или времени, относящиеся к годам с 100 по 9999 включительно. Длина поля 8 байт.

Счетчик (AutoNumber). Тип данных поля, в которое для каждой новой записи автоматически вводятся уникальные целые, последовательно возрастающие (на 1), или случайные числа. Значения этого поля нельзя изменить или удалить. Длина поля 4 байта для длинного целого, для кода репликации - 128 байт. По умолчанию в поле вводятся последовательные значения. В таблице не может быть более одного поля этого типа. Используется для определения уникального ключа таблицы

Логический (Yes/No). Логические данные, которые могут иметь одно из двух возможных значений Да/Нет; Истина/Ложь; Вкл./Выкл. (Yes/No; True/False; On/Off). Длина поля 1 бит

Поле объекта OLE (OLE Object). Объект (например, электронная таблица Microsoft Exce1, документ Microsoft Word, рисунок; звукозапись или другие данные в двоичном формате), связанный или внедренный в таблицу Access. Длина поля - до 1 Гигабайта (ограничивается объемом диска). Для полей типа OLE и MEMO не допускается сортировка и индексирование

Гиперссылка (Hyperlink). В качестве гиперссылки можно указывать путь к файлу на жестком диске, путь UNC или адрес URL. Если щелкнуть мышью на поле гиперссылки, Access выполнит переход на соответствующий объект, документ, страницу Web или другое место назначения. Максимальная длина 64 000 символов

Мастер подстановок...(Lookup Wizard...). Выбор этого типа данных запускает мастера подстановок. Мастер строит для поля список значений на основе полей из другой таблицы. Значения в такое поле будут вводиться из одного из полей списка. Соответственно, фактически тип данных поля определяется типом данных поля списка. Возможно также определение поля со списком постоянных значений.

Общие свойства

Общие свойства задаются для каждого поля на вкладке Общие (General) и зависят от выбранного типа данных.

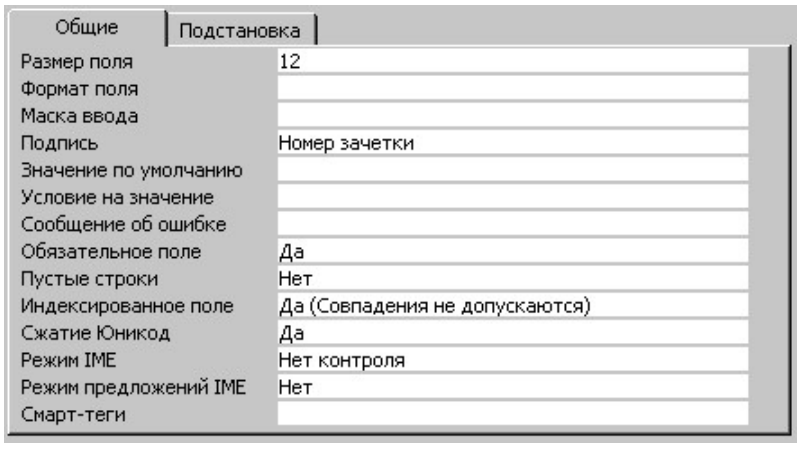


Рисунок 3.5 – Настройка свойств поля в окне Конструктора таблиц

Для отображения свойств поля необходимо установить курсор на строке соответствующего поля:

Размер поля (FieldSize) задает максимальный размер данных, сохраняемых в поле. Для поля с типом данных Текстовый задается размер от 1 до 255 байтов (по умолчанию 50 байт)

Для поля с типом данных Счетчик можно задать:

- Длинное Целое (Long Integer) - 4 байта
- Код репликации (Replication ID) - 128 байт

Для поля с типом данных Числовой можно задать:

- Байт (Byte) для целых чисел от 0 до 255, длина поля 1 байт
- Целое (Integer) для целых чисел от -32.768 до + 32.767, занимает 2 байта
- Длинное целое (Long Integer) для целых чисел от -2.147.483.648 до +2.147.483.647, занимает 4 байта
- Дробные с плавающей точкой 4 байта (Single) для чисел от -3,4х1038 до + 3,4хЮ38 с точностью до 7 знаков
- Дробные с плавающей точкой 8 байт (Double) для чисел от -1,797хЮ308 до +1,797хЮ308 с точностью до 15 знаков
- Действительное (Decimal) для целых чисел от -1038-1 до 1038-1 (при работе с проектами, которые хранятся в файлах типа .abp) и от -1028-1 до 1028-1 (.mdb) с точностью до 28 знаков, занимает 12 байт
- Код репликации. Глобальный уникальный идентификатор (Globally unique identifier, GUID), занимает 16 байт. Поля такого типа используются Access для создания системных уникальных идентификаторов реплик, наборов реплик, таблиц, записей и других объектов при репликации баз данных.

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

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

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

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

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

Число десятичных знаков (DecimalPlaces) задает для числового и денежного типов данных число знаков после запятой. Можно задать число от 0 до 15. По умолчанию (значение Авто, Autto) это число определяется установкой в свойстве Формат поля (Format). Следует иметь в виду, что установка этого свойства не действует, если свойство Формат поля (Format) не установлено или выбрано значение Основной (General Number). Свойство Число десятичных знаков (Decimal Places) влияет только на количество десятичных знаков, отображаемых на экране, и не влияет на число сохраняемых десятичных знаков. Для изменения числа сохраняемых знаков: нужно изменить свойство Размер поля (FieldSize)

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

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

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

Тип элемента управления

На вкладке Подстановка (Lookup) в окне конструктора таблиц задается свойство Тип элемента управления (DisplayControl).

Рисунок 3.6 – Закладка определения подстановки Конструктора таблиц

Это свойство определяет, будет ли отображаться поле в таблице и в форме в виде Поля (Text Box), Списка (List Box) или Поля со списком (Combo Box). Таким образом определяется вид элемента управления, используемого по умолчанию для отображения поля.

Если для поля выбран тип элемента управления Список (List Box) или Поле со списком (Combo Box), на вкладке Подстановка (Lookup) появляются дополнительные свойства, которые определяют источник данных для строк списка и ряд других характеристик списка. В качестве источника данных для списка выбирается таблица, с которой осуществляется постоянная связь, что обеспечивает актуальное состояние списка.

Определение первичного ключа

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

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

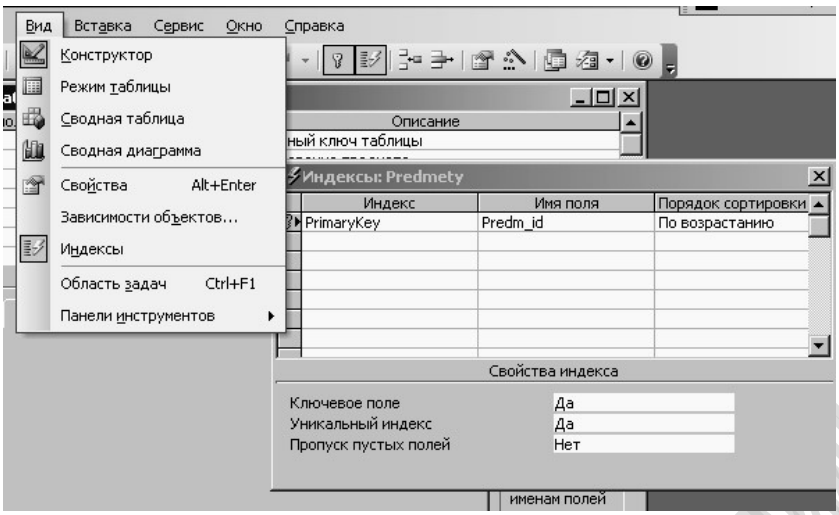


Рисунок 3.7 – Диалог определения первичного ключа таблицы

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

После определения структуры таблицы ее надо сохранить. Для этого используется команда Файл|Сохранить (File|Save) или кнопка панели инструментов конструктора Сохранить (Save). В окне Сохранение (Save As) вводится имя таблицы.

При сохранении таблицы происходит обновление файла базы данных, в которую помещается созданная таблица.

После сохранения таблицы становится доступным режим, позволяющий перейти ко второму этапу создания таблицы - созданию записей. Переход в этот режим, называемый режимом таблицы, возможен только после сохранения структуры таблицы и осуществляется нажатием кнопки Вид (View) на панели инструментов конструктора таблиц или выбором этого режима при открытии списка на этой кнопке.

В СУБД Access процесс создания реляционной базы данных включает создание схемы данных. Схема данных наглядно отображает таблицы и связи между ними, а также обеспечивает использование связей при обработке данных. В схеме данных устанавливаются параметры обеспечения целостности связей в базе данных.

Таким образом, осуществляется неразрывная связь внемашинного проектирования базы данных с этапом ее создания с помощью СУБД. В схеме данных, построенной по нормализованной модели данных предметной области, могут быть установлены одно-однозначные и одно-многозначные связи. Для таких связей обеспечивается поддержание целостности взаимосвязанных данных, при которой не допускается наличия в базе данных подчиненной записи без связанной с ней главной, при первоначальной загрузке базы данных и ее корректировках. Связи, определенные в схеме данных, используются автоматически при разработке многотабличных форм, запросов, отчетов, существенно упрощая процесс их конструирования.

При создании в Access схемы данных в ней определяются и запоминаются связи между таблицами. Это позволяет системе автоматически использовать связи, один раз определенные в схеме данных, при создании форм, запросов, отчетов на основе взаимосвязанных таблиц, а пользователь освобождается от необходимости указывать эти связи при конструировании этих

объектов. Схема данных базы графически отображается в своем окне, где таблицы представлены списками полей, а связи - линиями между полями разных таблиц.

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

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

Включение таблиц в схему данных. После нажатия кнопки Схема данных (Relationships) открывается окно Добавление таблицы (Show Table) , в котором можно выбрать таблицы и запросы, включаемые в схему данных. Для размещения таблицы в окне Схема данных (Relationships) надо выделить ее в окне Добавление таблицы (Show Table) и нажать кнопку Добавить (Add). Для выделения нескольких таблиц надо, удерживая клавишу , щелкнуть мышью на каждой из этих таблиц. Включив все нужные таблицы в схему данных, нажать кнопку Закрыть (Close).
В результате в окне Схема данных (Relationships) будут представлены все включенные таблицы со списком своих полей. Далее можно приступать к определению связей между ними.

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

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

При создании связи по составному ключу необходимо выделить все поля, входящие в ключ главной таблицы, и перетащить их на одно из полей связи в подчиненной таблице. Для выделения всех полей, входящих в составной уникальный ключ, необходимо отмечать поля при нажатой клавише . После создания связи откроется окно Изменение связей (Edit Relationships). При этом в строке Тип отношения (Relationship Type) автоматически установится тип один-ко-многим (One-To-Many).

При составном ключе связи в окне Изменение связей (Edit Relationships) необходимо для каждого поля ключа в главной таблице ТАБЛИЦА/ЗАПРОС (Table/Query) выбрать соответствующее поле подчиненной таблицы, названной СВЯЗАННАЯ ТАБЛИЦА/ЗАПРОС (Related Table/Query).

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

Обеспечение связной целостности данных означает, что Access при корректировке базы данных обеспечивает для связанных таблиц контроль за соблюдением следующих условий:

- в подчиненную таблицу не может быть добавлена запись с несуществующим в главной таблице значением ключа связи;

– в главной таблице нельзя удалить запись, если не удалены связанные с ней записи в подчиненной таблице;
– изменение значений ключа связи в записи главной таблицы невозможно, если в подчиненной таблице имеются связанные с ней записи.

При попытке пользователя нарушить эти условия в операциях добавления и удаления записей или обновления ключевых данных в связанных таблицах Access выводит соответствующее сообщение и не допускает выполнения операции.

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

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

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

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

В режиме каскадного обновления связанных полей при изменении значения поля связи в записи главной таблицы, Access автоматически изменит значения в соответствующем поле в подчиненных записях.

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

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

Для изменения типа связи между таблицами или контроля целостности данных необходимо, находясь в схеме данных, щелкнуть правой кнопкой мыши по линии, отображающей связь и выбрать пункт контекстного меню «Изменить связь».

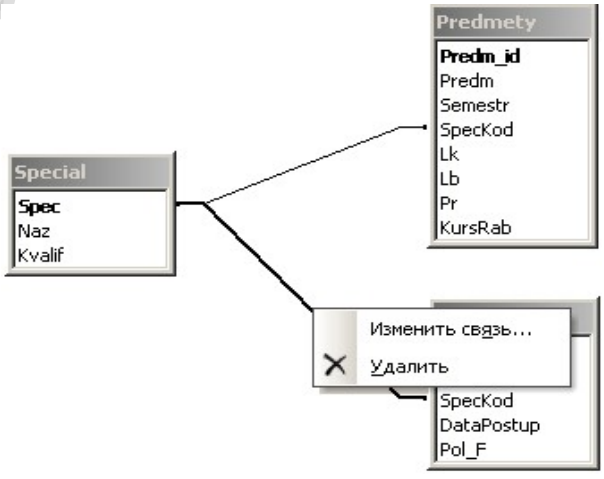


Рисунок 3.8 – Изменение связи между таблицами в схеме данных

В результате появится диалог «Изменение связи», который позволяет точно определить все необходимые параметры:

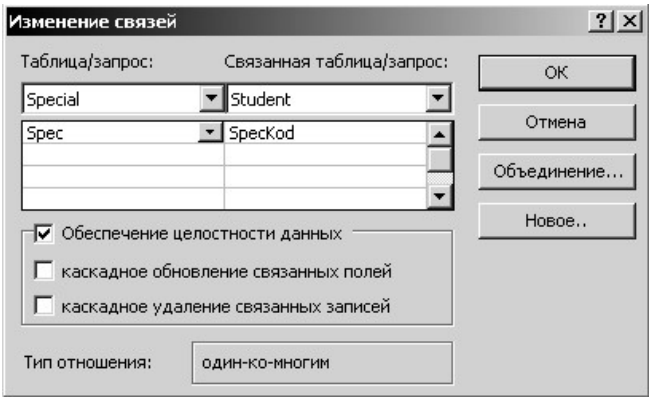


Рисунок 3.9 – Диалог изменения параметров связи между таблицами

Объем базы данных может быть очень велик. В большинстве случаев для решения конкретных задач требуется лишь часть хранящейся в базе информации. Работа с данными значительно упростится, если исключить из просмотра записи, несущественные с точки зрения поставленной задачи. При работе в Microsoft Access 2000 в интерактивном режиме этой цели служат такие инструменты, как сортировка и фильтры.

Сортировка записей

Сортировка записей в порядке возрастания или убывания значений одного поля (поля сортировки) может быть выполнена как в режиме таблицы, так и в режиме формы. Для этого следует установить курсор в поле сортировки и нажать соответствующую кнопку панели инструментов Таблица в режиме таблицы (Table Datasheet):

или

Сортировка записей может быть выполнена также командой меню Записи|Сортировка (Records|Sort).

Для сортировки записей таблицы по нескольким полям необходимо выделить соответствующие столбцы и выполнить команду сортировки. Сортируемые поля должны быть смежными, старшим полем сортировки является поле, расположенное слева.

Использование фильтра для отбора данных

Для просмотра и корректировки записей базы данных, удовлетворяющих указанным пользователем условиям отбора, предусмотрена фильтрация таблицы. Фильтр - это набор условий, применяемый для отбора подмножества записей из таблицы, формы или запроса.

Фильтр по выделенному. Простейшим способом задания условия отбора записей является выделение в таблице или форме некоторого значения поля или его части.

Функция Фильтр по выделенному (Filter By Selection) извлекает из таблицы и выдает на экран только те записи, которые содержат выделенное значение. Это относительно простой фильтр, допускающий одно условие на поле.

Чтобы задать фильтр по нескольким полям, нужно выделить эти поля, а затем щелкнуть на кнопке Фильтр по выделенному (Filter By Selection).

Кроме фильтра по выделенному можно использовать и другие два типа фильтров: обычный и расширенный.

Обычный фильтр. После выполнения команды Записи|Фильтр|Изменить фильтр (Records| Filter| Filter By Form) в окне обычного фильтра Фильтр (Filter By Form) на экран выводится пустая таблица или форма для активного объекта базы данных. На вкладке Найти (Look For) в поля фильтра вводятся значения, по которым будут отбираться записи. Ввод значений в несколько полей одной строки фильтра определяет отбор записей, в которых присутствуют все указанные значения. При этом заданные условия рассматриваются как объединяемые логической операцией "И".

Для того чтобы указать альтернативный вариант отбора записей, включаемых в результирующий набор, выбирается вкладка Или (Or) в левом нижнем углу окна фильтра. При этом открывается новая строка фильтра для задания альтернативного варианта. Набор условий, введенных в разные строки, связывается логической операцией "ИЛИ". Следующий альтернативный вариант может быть задан на следующей вкладке Или (Or).

Фильтр будет отбирать записи, содержащие в полях все значения, указанные на вкладке Найти (Look For), и записи, содержащие в полях все значения, указанные на вкладке Или (Or).

Для выполнения фильтрации надо нажать кнопку
Применение фильтра (App1y Filter).

Расширенный фильтр. После выполнения команды Записи|Фильтр|Расширенный фильтр (Records| Filter| Advanced/Sort...) в окне расширенного фильтра в верхней части выводится список полей активной таблицы. В нижней части окна выводится бланк запроса. В строку бланка запроса Поле (Fild) перетаскиваются мышью из списка поля, по которым надо задать условия отбора записей. Условия отбора вводятся в соответствующую строку. Кроме того, в бланке запроса в строке Сортировка (Sort) может быть выбран тип сортировки для одного или нескольких выбранных полей.

Сохранение фильтра. Обычный и расширенный фильтры так же, как и фильтр по выделенному, сохраняются автоматически при сохранении таблицы, запроса или формы. Этот сохраненный фильтр при повторных открытиях объекта является текущим и может использоваться по команде Записи|Применение фильтра (Records|Apply Filter/Sort). Если создается новый фильтр, он заменяет любой фильтр, ранее сохраненный с формой или объектом в режиме таблицы.

Для уничтожения фильтра надо на панели инструментов Фильтр (Filter/Sort) нажать кнопку «Очистить бланк» (Clear Grid) и затем кнопку «Применение фильтра» (Apply Filter). Только при этом фильтр, сохраняемый с таблицей, уничтожается.

Сохранение группы фильтров. Целесообразно подготовить и сохранить несколько фильтров для таблицы или формы, которые в дальнейшем могут быть загружены в окно фильтра и применены. Для сохранения подготовленного фильтра нужно выполнить команду Записи|Фильтр|Изменить фильтр (Records| Filter| Filter By Form) или нажать соответствующую кнопку панели инструментов и затем выполнить команду Файл|Сохранить как запрос (File|Save As Query) или нажать соответствующую кнопку панели инструментов Фильтр (Filter). Для выполнения команды можно также при активном окне фильтра вызвать контекстное меню. В открывшемся диалоговом окне Сохранение в виде запроса (Save As Query) надо ввести имя сохраняемого фильтра в поле Имя запроса (Query Name).

В дальнейшем, вместо того чтобы создавать фильтр заново, можно использовать существующий фильтр. Для этого нужно перейти в режим Изменить фильтр (Filter By Form) и выполнить команду Файл|Загрузить из запроса (File|Load From Query). В окне Применяемый фильтр (Applicable Filter) выбирается нужный фильтр. Заметим, что в списке отображаются (помимо сохраненных ранее фильтров) запросы, созданные на основе фильтруемой таблицы.

Как упоминалось ранее, в любой СУБД информация хранится в таблицах, в которых ее можно и посматривать, и даже до некоторой степени управлять ею посредством некоторого набора универсальных фильтров. Однако для повседневной работы этого слишком мало, особенно в случаях, когда заранее неизвестно, есть ли в базе данных интересующая

информация или нет. Для того чтобы пользователи (а если быть более точными - разработчики баз данных) могли свободно манипулировать информацией, в СУБД был встроен механизм запросов, предназначенный для тонкого управления содержимым любой базы данных.

Все запросы делятся на запрос-выборку и запрос-действие.

Запрос-выборка просто генерирует ответ на заданный пользователем вопрос и на этом заканчивает свою работу. После выполнения запроса СУБД создает виртуальную таблицу, в которую заносит выбранную информацию и хранит ее до тех пор, пока сгенерированная таблица не будет закрыта. Фактически, механизм запроса-выборки работает следующим образом: до тех пор, пока конкретный запрос не инициализирован, он представляет собой всего лишь безжизненный набор каких-то инструкций, которые к тому же могут оказаться еще и неправильными. Когда пользователь или заранее написанный модуль обращается к конкретному запросу, его набор инструкций немедленно выполняется, в оперативной памяти компьютера возникает виртуальная итоговая таблица, которая, в свою очередь, сама может служить источником данных для другого запроса или пользовательской формы. Когда этот запрос закрывается, полученная таблица уничтожается, освобождая занимаемую память.

В отличие от запроса-выборки, запрос-действие оставляет после себя нечто конкретное и ощутимое. При помощи запроса-действия можно автоматически создать новую таблицу, внести данные в уже имеющуюся таблицу, а также удалить из существующей таблицы какой-либо набор записей.

Во избежание совершения непреднамеренных действий с данными, которые нельзя отменить, разработчики СУБД Microsoft Access 2000 настроили систему таким образом, что любой создаваемый запрос автоматически считается запросом-выборкой и работает в соответствии с этим. В том случае, когда пользователю или разработчику нужно получить запрос-действие, системе требуется об этом "сказать" отдельно.

Механизм запросов является самым универсальным решением, позволяющим, с одной стороны, как угодно манипулировать данными, вплоть до того, что в зависимости от своего состояния база данных выбирала какие-то определенные действия, а с другой - снизить потребность СУБД в системных ресурсах вообще, так как активизация базы данных не приводит к проведению всех имеющихся в ней запросов и прочих автоматических функций, как это имеет место, например, в электронных таблицах. Те или иные запросы выполняются компьютером только тогда, когда это действительно необходимо. Правда, медаль имеет и оборотную сторону, исходя из своей идеологии запрос чрезвычайно сильно похож на моментальное фото, отражающее текущее состояние базы СТРОГО на момент выполнения данного запроса. Иными словами, если в данных произошли перемены ПОСЛЕ того, как запрос закончил свою работу, то указанные изменения в уже созданной выборке отражены не будут, так как виртуальная таблица с самими данными никакой связи не имеет. Это означает, что, во-первых, при проектировании любой базы данных следует обращать особое внимание на очередность запуска конкретных запросов, а во-вторых, если по каким-либо причинам пользователь нуждается в постоянном мониторинге в реальном режиме времени, то единственным способом достичь этого является повторение одного и того же запроса через короткие промежутки времени.

Для того чтобы создать запрос-выборку, нужно перейти на вкладку ЗАПРОСЫ и нажать экранную кнопку " СОЗДАТЬ". Это приведет к появлению мастера запросов, который желает знать, каким именно алгоритмом разработчик желает воспользоваться: одним из типовых или специализированным конструктором, в котором буквально весь запрос создается пользователем вручную.

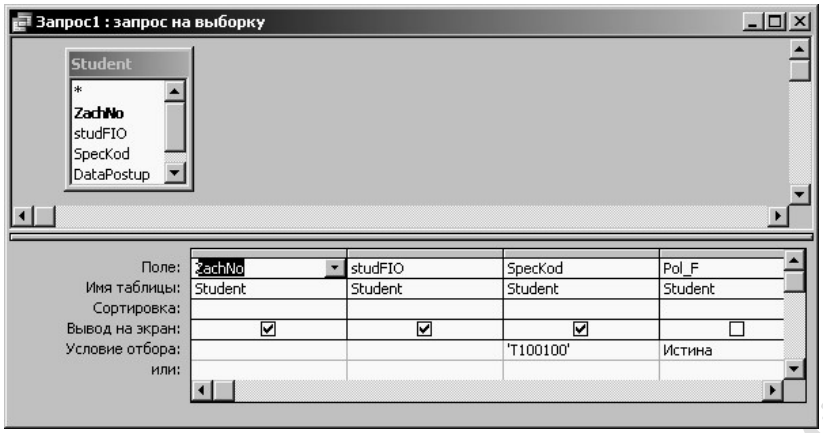


Рисунок 3.10 – Создание запроса при помощи Конструктора

Тут уместно вспомнить, что разработчики Microsoft Access заранее позаботились о том, чтобы в типовых ситуациях пользователь мог воспользоваться стандартным видом запроса. В том случае, когда пользователь желает выбрать некие данные только из одной таблицы, то лучше всего воспользоваться вариантом " Простой запрос". В том случае, если пользователю необходимо нечто близкое к итоговому отчету, например, из всего объема продаж товаров и услуг за всю историю предприятия необходимо выбрать показатели по конкретным позициям за конкретный год, несколько определенных месяцев или кварталов, следует воспользоваться вариантом " Перекрестный вопрос". В том случае, когда из всего массива данных одной или нескольких таблиц нужно выбрать лишь те данные, которые повторяются, например, выбрать те марки автомобилей, количество продаж которых "больше чем один", следует воспользоваться вариантом " Повторяющиеся записи". В том случае, когда нужно найти среди всех записей базы данных те, которые не имеют подчиненных, следует выбирать последний вариант, " Записи без подчиненных".

Диалоговое окно конструктора запросов все же не дает возможностей воспользоваться всеми средствами языка SQL. Разработчики MS Access предусмотрели возможность работы непосредственно с текстом SQL-запроса: для этого надо, находясь в режиме конструктора, выбрать из меню «Вид | Режим SQL»:

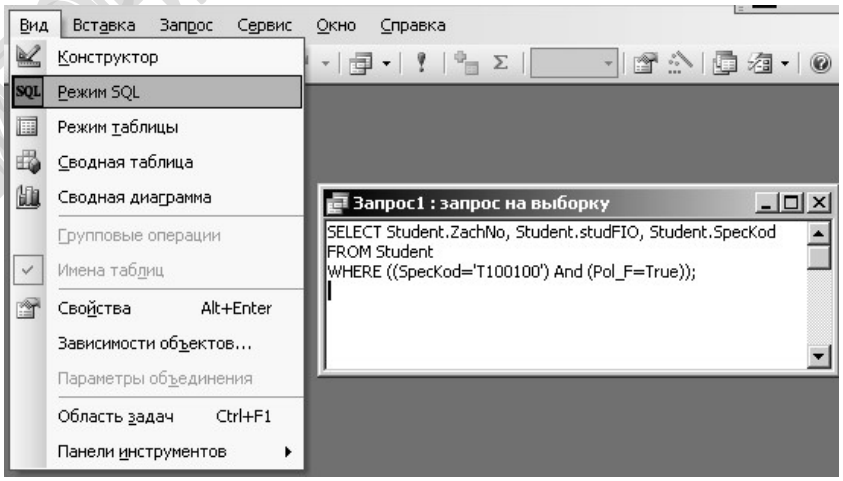


Рисунок 3.11 – Переход в SQL-режим создания запросов

После создания таблиц и запросов можно организовать ввод данных с помощью формы. Иногда пользователи предпочитают работать с данными, выведенными в виде таблицы, которая напоминает им одну из рабочих книг. Но часто бывает удобнее организовать данные в виде формы ввода, когда видно только одну запись, поля которой расположены в нужном порядке.

При этом вы можете в полной мере проявить здесь свои способности к дизайнерскому искусству, если таковые имеются.

При желании создать простейшую форму воспользуйтесь опцией автоформа. Это самый простой способ создания формы. Вы получаете форму с набором текстовых полей, перед которыми выводятся названия или заголовки полей, если последние имеются для поля. В дальнейшем, по мере освоения пакета, вы сможете использовать эту возможность для создания основы вашей формы с целью дальнейшего ее усовершенствования.

Если вы решите создавать новую форму, воспользовавшись кнопкой «Создать» на вкладке Формы, то в списке доступных режимов создания будут присутствовать автоформы, причем трех типов: состоящая из колонок (ее вы можете построить предыдущим способом), ленточная и табличная.

Вариант В СТОЛБЕЦ должен быть весьма привычен тем, кто пользовался режимом базы данных в электронных таблицах Microsoft Excel. В данном случае все поля формы располагаются друг над другом, и в один конкретный момент времени на экране видны данные только одной из записей таблицы базы данных.

Вариант ЛЕНТОЧНАЯ больше подходит тогда, когда опорная таблица хранит некоторые значения одного параметра. Например, ленточная форма прекрасно отражает продажи по датам. В одном столбце можно вывести значения дат, а во втором - объем соответствующих им продаж. Наглядно и удобно.

Вариант ТАБЛИЧНАЯ, как следует уже из самого наименования, предназначен для автоматического создания форм, внешне похожих на таблицы. В том случае, если, вместо конкретных цифровых величин, вам нужно показать пользователю более наглядную картинку, то следует использовать вариант ДИАГРАММА. В этом случае, вместо цифр и ячеек, Microsoft Excel автоматически строит диаграмму по указанным полям, под вашим руководством "форматирует" ее и выводит на экран.

Ну, и напоследок, для тех случаев, когда возникает необходимость отображать не излишнюю конкретику, а некий интегрированный суммарный результат, вам идеально подойдет вариант СВОДНАЯ ТАБЛИЦА.

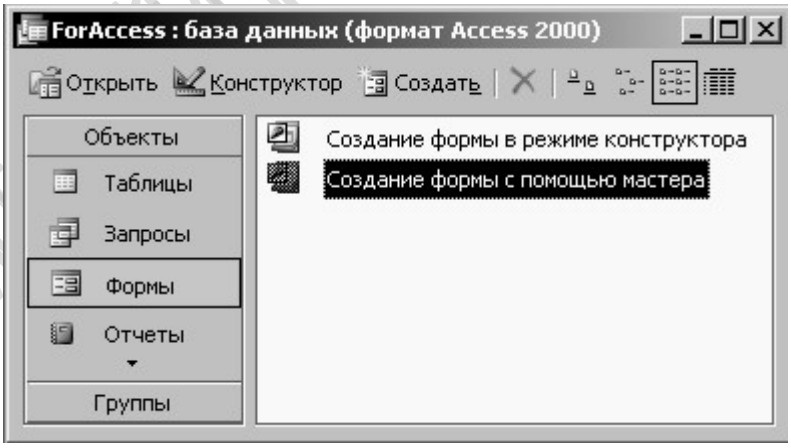


Рисунок 3.12 – Раздел форм основного окна разработки базы данных

После того как вы решите создать форму, перед вами появится диалоговое окно Новая форма, в котором имеется целых семь опций. Обращаем ваше внимание, что надо выбрать источник данных, в противном случае для вывода данных надо применять более сложные технологии. Можете выбрать любой способ, но следует отметить, что на самом деле это не семь способов создания форм.

К примеру, опция Простая форма, которая запускает Мастер создания форм, по своим возможностям раскладывается на три следующие за ними автоформы, но с предоставлением

возможности выбрать из исходных таблиц нужные вам поля, задать свой заголовок, выбрать один из стилей оформления.

Формы, которые удовлетворят любому, даже самому требовательному вкусу, можно создать с помощью Конструктора форм. Эффективным способом работы является быстрый выбор полей с помощью Мастера создания форм, стиля форм и дальнейшее совершенствование форм с помощью Конструктора.

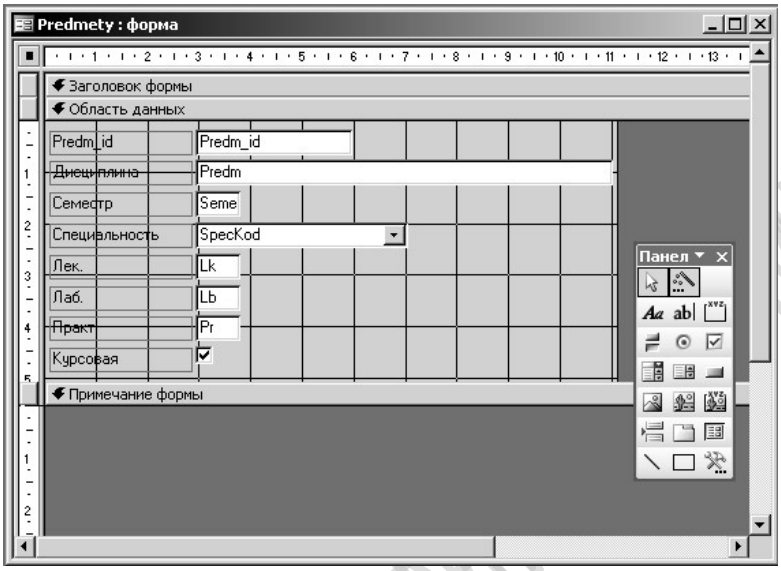


Рисунок 3.13 – Окно Конструктора формы

Любая форма Access состоит из трех важных областей: *заголовок, область данных и область примечаний*. Но, по умолчанию Access создает новую форму, состоящую только из области данных, как это видно на предыдущем рисунке. В этом кроется важный смысл. Дело в том, что каждая из областей имеет свое собственное назначение, что позволяет создавать самые функционально разные формы. Область заголовка существует для тех элементов, которые носят общий характер. Например, это могут быть наименования столбцов в ленточных и табличных формах. Названия самих форм также помещаются в область заголовка. Область данных предназначена для размещения на ней самой представляемой информации. Информация может выводиться в обычные поля, в поля со списком, в виде переключателей и так далее. Область примечаний существует для вывода всяческих пояснений и подсказок, если такое будет признано необходимым. Обычно не так уж часто требуется предусматривать в формах дополнительные пояснения. Как правило, достаточно подобрать информативные названия полей, но случаи бывают разные, и иногда без специальных комментариев просто не обойтись. Словом, в зависимости от обстоятельств вы можете применять только область данных, а можете перевести форму в полное представление.

Это делается при помощи контекстно-зависимого меню мыши, выбирая режим «Заголовок/примечание формы».

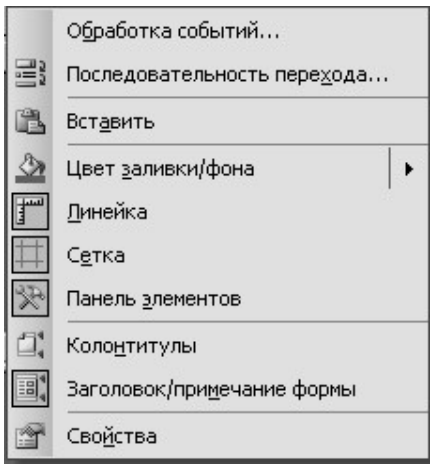


Рисунок 3.14 – Контекстное меню Конструктора формы

Разрабатывая форму, следует помнить, что все ее элементы делятся на две большие группы. Первая, это, например, информационные надписи, является полностью автономной и ни с каким источником данных не связана. Вторая, например, поля ввода или переключатели, без связи с таблицей вообще теряет всякий смысл. Это определяет всю методику работы с указанными элементами.

Создание форм отнимает больше половины времени, затрачиваемого на создание приложений. Но именно на этом этапе работы вы можете воспользоваться наибольшим количеством средств автоматизации. Вам предоставляется большое количество встроенных объектов. Со многими объектами связаны Построители (Мастера, Wizards), причем число их разновидностей так велико, что позволяет построить автоматизировано 90%, а в некоторых случаях и целиком приложение.

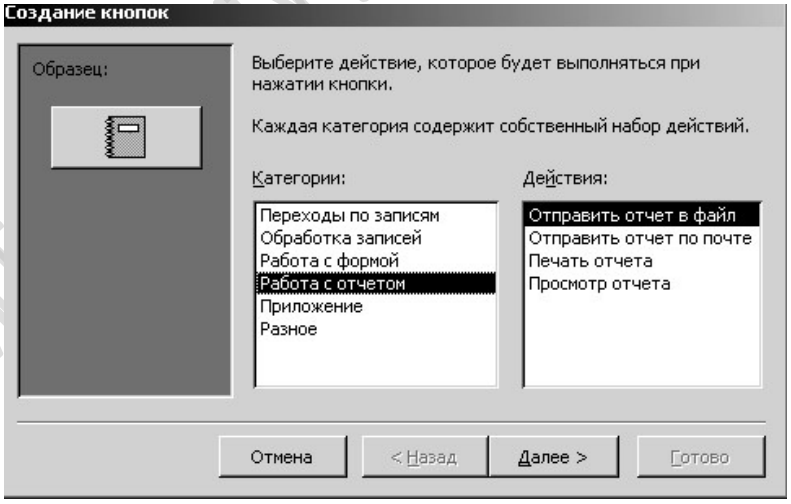


Рисунок 3.15 – Диалоговое окно Мастера кнопок

Помимо кнопок, Мастера имеются для Групп, причем вы сами можете решить, какие объекты поместить внутрь контейнера группы, для списков, комбинированных списков.

Каждый объект имеет большой набор свойств и событий. Событию можно присвоить макрокоманду или процедуру, которые будут вызываться при его наступлении. С помощью этого можно добиться значительной гибкости работы с формой.

Для форм доступны три режима работы: Конструктор, Форма и Таблица. Режим вывода данных имеет три вида: ленточная форма, простая форма и таблица. При работе с простой

формой одновременно вы можете видеть данные только из одной записи, при ленточной - одну и более, в зависимости от того, сколько можно уместить их на экран.

Используя установки, которые вам доступны после выбора команды Параметры из меню Сервис, вы можете задать шаблон формы, в качестве которого может использоваться любая заранее созданная форма. Все новые формы будут создаваться на основе этой формы со всеми включенными в нее элементами управления и свойствами.

Формы и элементы управления можно создавать и модифицировать программно, но занимает это намного больше времени.

С формой также удобно работать, как и с самой таблицей, а нередко даже удобнее. Особенно это касается сложных форм, которые позволяют вывести на экран несколько взаимосвязанных записей одновременно, что совершенно не под силу ни одной таблице. Кроме того, механизм форм позволяет прятать от конечного пользователя базы данных все лишнее и в данный момент ненужное, лишь отвлекающее внимание от основной работы. Да и по наглядности представления информации с формой не может тягаться ни одна таблица.

По аналогии с таблицами и запросами, все формы, простые и сложные, словом абсолютно все, хранятся в Microsoft Access на вкладке ФОРМЫ, что облегчает их поиск. Форма это точно такой же объект СУБД, как запрос или таблица, а, значит, ее можно вызывать при помощи макроса или модуля, как того потребует рабочая необходимость. Единственное, что не предусмотрели разработчики СУБД MS Access, так это механизм автоматического связывания форм и задания их иерархии. Об этом разработчику придется позаботиться самостоятельно.

Отчеты наряду с формами создают представление о вашем приложении, и создание их обычно требует кропотливого труда. Компания Microsoft постаралась ваш труд облегчить. Для каждой таблицы вы можете создать Автоотчет. При этом Автоотчет, доступный для создания с помощью меню или кнопки Новый объект на панели инструментов База данных, создает отчет, данные в котором будут выведены в столбец. Еще один Автоотчет станет доступным при выборе кнопки Создать на вкладке Отчеты. Это ленточный автоотчет, когда данные из всех полей будут выводиться в колонку. Если вы хотите выбрать поля для отчета, а не выводить все, имеющиеся в таблице или запросе, то воспользуйтесь Мастером отчетов. Мастер отчетов позволяет, помимо выбора полей для отчета, сгруппировать данные по какому-нибудь полю, при этом вы можете установить интервал группировки, установить порядок сортировки, выбрать макет отчета и его стиль.

Среди прочих мастеров отметим Мастер по созданию отчета с диаграммой и построитель почтовых наклеек. Построитель почтовых наклеек - это мощное средство с разнообразными свойствами, требующее подробного изучения. Построитель почтовых наклеек по своему классу скорее можно отнести к средствам визуального проектирования.

Для построения сложных отчетов предназначен Конструктор отчетов. При его запуске вместе с ним загружается панель инструментов с элементами управления, которые можно размещать в различных областях проектируемого отчета путем буксировки с помощью мыши. Перед печатью отчета его можно просмотреть в окне предварительного просмотра.

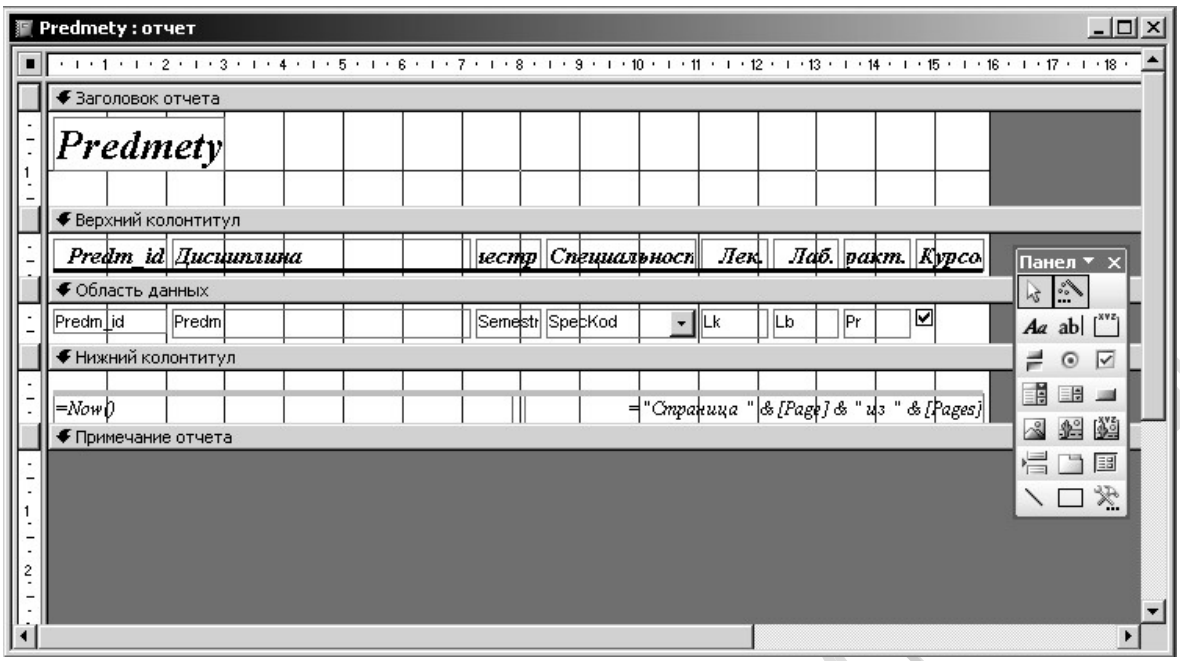


Рисунок 3.16 – Окно Конструктора отчетов

Как и многие другие объекты базы данных, отчеты можно создавать программно. Обычно этот метод используется для создания собственных Мастеров.

Макрокоманды, которые можно объединять в макросы, совершают разнообразные действия, выполнимые в СУБД Access, а с помощью параметров этим действиям можно придать гибкость, которой иначе можно добиться только путем кропотливого программирования. В Access имеется около пятидесяти макрокоманд. Для того чтобы создать макрос, вам обязательно нужно загрузить окно Конструктора макросов, которое состоит из двух частей.

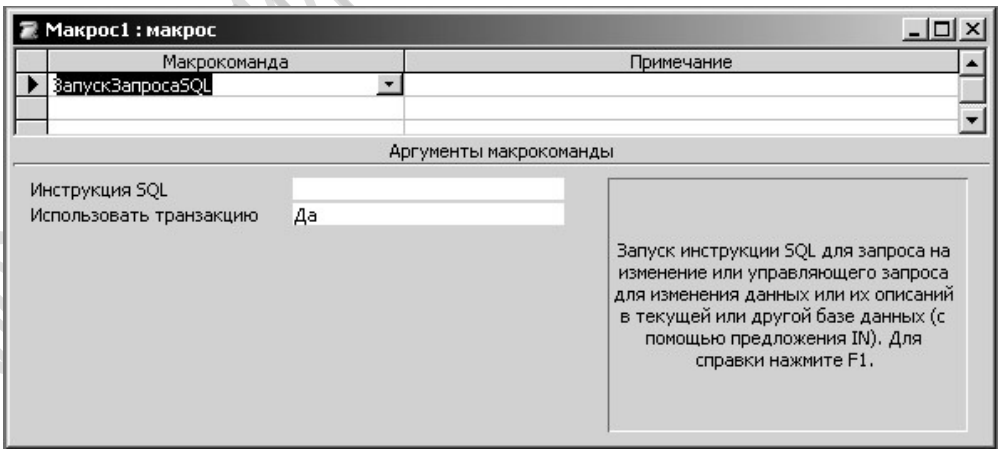


Рисунок 3.17 – Мастер построения макросов

Верхняя служит для ввода макрокоманд и представляет собой таблицу, имеющую от двух до четырех колонок, в зависимости от установленных параметров. В нижней части окна Конструктора макросов вводятся аргументы макроса. В зависимости от выбранной макрокоманды число аргументов динамически меняется. Напротив каждого аргумента имеется его описание. Справа отображается пояснение для рода и типа требуемого аргумента. Очень часто аргумент можно выбрать из раскрывающегося списка.

Для построения более сложных макросов, требующих выполнения определенных условий для запуска отдельных макрокоманд, необходимо добавить колонку Условия. После этого макрокоманды, находящиеся в одной строке с условием, будут выполняться, только если выражение условия будет истинным. Для того чтобы распространить действие условия на последующие строки, достаточно ввести многоточие (...) в колонке условия напротив соответствующих макрокоманд.

Access довольно высоким для настольного приложения СУБД уровнем встроенной системой защиты. Вы можете создавать группы, пользователей, присваивать права доступа ко всем объектам, в том числе и модулям. Кстати, это решает вопрос закрытия ваших процедур и функций от чужих глаз. Так как для Access нет компилятора, то необходимость защиты становится очень актуальной для разработчиков. Система защиты доступна только при открытой базе данных. Каждому пользователю можно предоставить индивидуальный пароль. Система защиты доступна как с помощью визуальных средств, так и программным путем. Если вы хотите защитить вашу базу данных даже от пользователя с именем Admin, то пользуйтесь услугами надстройки Security, которая поставляется вместе с Access Developer Toolkit. Помимо этого вы можете закрыть вашу базу данных от просмотра внешними программами.

Вопросы для самоконтроля:

- какие вы знаете способы создания таблиц в Access 2000;
- как использовать маску ввода и формата поля при определении полей таблиц;
- приведите основные особенности применения полей подстановки;
-проиллюстрируйте параметры связи между таблицами. Особенности задания параметров объединения.
- какие вы знаете средства автоматизированного построения форм и отчетов в Access
-приведите основные возможности модификации формы, предоставляемые режимом «Конструктор», визуальные объекты формы
-основные способы конструирования отчета, структура шаблона отчета и его модификация в режиме конструктора отчетов.

ЛАБОРАТОРНАЯ РАБОТА 2.1
СОЗДАНИЕ СХЕМЫ ДАННЫХ

Цель: научиться создавать в интерактивном режиме схему данных в СУБД MS Access 2000 с помощью Конструкторов.

Материалы и оборудование: ПК

Индивидуальное задание: создать при помощи Конструкторов в СУБД Access схему БД, спроектированную при выполнении лабораторной работы № 1, заполнить каждую таблицу 5-10 записями.

Требования к содержанию отчета

Отчет должен содержать: наименование работы, цель работы, постановку задачи, описание варианта задания, краткие теоретические сведения, описание хода выполнения лабораторной работы, результаты выполнения задания.

В качестве описания хода выполнения лабораторной работы и результата работы должны быть приведены:

- снимок с экрана (скриншот) схемы данных;

- описание структуры каждой таблицы и распечатки содержимого обеих таблиц, полученные с помощью возможностей связи MS Access с другими приложениями Office (Меню «Сервис»-> «Связи с Office»…)

Варианты заданий

Варианты заданий приведены в теме 1. При выполнении лабораторной работы следует создать в Access те таблицы, которые были разработаны при выполнении нормализации в ходе лабораторной работы 1.1.

ЛАБОРАТОРНАЯ РАБОТА 2.2
ПРОЕКТИРОВАНИЕ ФОРМ И ОТЧЕТОВ

Цель: ознакомится с применением мастеров и конструкторов системы Access для создания экранных форм и отчетов.

Материалы и оборудование: ПК

Индивидуальное задание: создать в СУБД Access базу данных из нескольких связанных таблиц. Заполнить каждую таблицу **10-15** записями и построить средствами СУБД несколько запросов по базе. Сконструировать экранные формы для всех таблиц базы и отчеты по 2-3 запросам или таблицам, на форму главной таблицы добавить кнопки вызова других форм и отчетов.

Требования к содержанию отчета

Отчет должен содержать: наименование работы, цель работы, постановку задачи, описание варианта задания, краткие теоретические сведения, описание хода выполнения лабораторной работы, результаты выполнения задания.

В качестве описания хода выполнения лабораторной и результата работы должны быть приведены:

- снимки экрана (скриншоты) разработанных форм, иллюстрирующие процесс их конструирования и результат.
- распечатки текстов отчетов, спроектированных для базы.

Варианты заданий

В качестве основы для выполнения лабораторной работы 3.2 используются таблицы, разработанные при выполнении лабораторной работы 1.1 в соответствии с вариантом, прилагаемым к заданию 1.1.

ТЕМА 3 ЯЗЫК SQL И ЕГО ВОЗМОЖНОСТИ

Основные понятия по теме
1 Основные характеристики языка SQL.
2 Запрос выборки данных SELECT. Выбор из одной таблицы.
3 SQL-функции и группировки в запросе выборки данных.
4 Выборка данных из нескольких таблиц.
5 Модификация данных средствами SQL.
6 Безопасность и санкционирование доступа.

Язык SQL (Structured Query Language - структуризованный язык запросов) является непроцедурным языком и ориентирован на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц. Особенность предложений этого языка состоит в том, что они ориентированы в большей степени на конечный результат обработки данных, чем на процедуру этой обработки. SQL сам определяет, где находятся данные, какие индексы и даже наиболее эффективные последовательности операций следует использовать для их получения: не надо указывать эти детали в запросе к базе данных.

Появление теории реляционных баз данных и предложенного Коддом языка запросов "alpha", основанного на реляционном исчислении, инициировало разработку ряда языков запросов, которые можно отнести к двум классам:

– алгебраические языки, позволяющие выражать запросы средствами специализированных операторов, применяемых к отношениям (JOIN - соединить, INTERSECT - пересечь, SUBTRACT - вычесть и т.д.);

– языки исчисления предикатов представляют собой набор правил для записи выражения, определяющего новое отношение из заданной совокупности существующих отношений. Другими словами исчисление предикатов есть метод определения того отношения, которое нам желательно получить (как ответ на запрос) из отношений, уже имеющихся в базе данных.

Разработка, в основном, шла в отделениях фирмы IBM (языки ISBL, SQL, QBE) и университетах США (PIQUE, QUEL). Последний создавался для СУБД INGRES (Interactive Graphics and Retrieval System), которая была разработана в начале 70-х годов в Университете шт. Калифорния и сегодня входит в пятерку лучших профессиональных СУБД. Сегодня из всех этих языков полностью сохранились и развиваются QBE (Query-By-Example - запрос по образцу) и SQL, а из остальных взяты в расширение внутренних языков СУБД только наиболее интересные конструкции.

В начале 80-х годов SQL «победил» другие языки запросов и стал фактическим стандартом таких языков для профессиональных реляционных СУБД. В 1987 году он стал международным стандартом языка баз данных и начал внедряться во все распространенные СУБД персональных компьютеров.

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.

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.

В некоторых СУБД еще существует тип данных LOGICAL, DOUBLE и ряд других. СУБД INGRES предоставляет пользователю возможность самостоятельного определения новых типов данных, например, плоскостные или пространственные координаты, единицы различных метрик, пяти- или шестидневные недели (рабочая неделя, где сразу после пятницы или субботы следует понедельник), дроби, графика, большие целые числа (что стало очень актуальным для российских банков) и т.п.

Ориентированный на работу с таблицами SQL не имеет достаточных средств для создания сложных прикладных программ. Поэтому в разных СУБД он либо используется вместе с

языками программирования высокого уровня (например, такими как Си или Паскаль), либо включен в состав команд специально разработанного языка СУБД (язык систем dBASE, R:BASE и т.п.).

Унификация полных языков современных профессиональных СУБД достигается за счет внедрения объектно-ориентированного языка четвертого поколения 4GL. Последний позволяет организовывать циклы, условные предложения, меню, экранные формы, сложные запросы к базам данных с интерфейсом, ориентированным как на алфавитно-цифровые терминалы, так и на оконный графический интерфейс (X-Windows, MS-Windows).

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. Представляет исторический интерес тот факт, что именно возможность включения одного предложения SELECT внутрь другого послужила мотивировкой использования прилагательного "структуризированный" в названии языка SQL.

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 CREAT VIEW, DECLARE CURSOR или INSERT;

– средство присвоения глобальным переменным значений из строк сформированной таблицы (INTO-фраза).

Предложение SELECT (выбрать) имеет следующий формат:

```
SELECT [[ALL] | DISTINCT] { * | эл-т_выбора [,эл-т_выбора]…}
FROM {базовая_таблица | представление} [псевдоним]
     [,{базовая_таблица | представление} [псевдоним]] ...
[WHERE   условие]
[GROUP BY фраза [HAVING условие на группу]]
[ORDER BY {таблица.столбец | № эл-та_выбора } [[ASC] | DESC]
[,{таблица.столбец | № эл-та_выбора } [[ASC] | DESC]] ...;
```

Обозначает этот запрос следующее:

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

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

Элемент_выбора - это одна из следующих конструкций:

* | значение|SQL_функция | системная_переменная

где «значение» это:
таблица.столбец | (выражение) | константа | переменная

Фраза WHERE включает набор условий для отбора строк:

WHERE [NOT] условие [[AND|OR][NOT] условие1]...

где «условие» одна из следующих конструкций:

значение { = | <> | < | <= | > | >= } { значение | (подзапрос) }

значение_1 [NOT] BETWEEN значение_2 AND значение_3

значение [NOT] IN { (константа [,константа]...) | (подзапрос) }

значение IS [NOT] NULL

таблица.столбец [NOT] LIKE 'строка_символов' [ESCAPE 'символ']

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, OR, AND NOT, OR NOT), причем существует приоритет AND над OR (сначала выполняются все операции AND и только после этого операции OR).

Синтаксис фразы GROUP BY имеет вид:

GROUP BY таблица.столбец [,таблица.столбец] ... [HAVING условие]

GROUP BY инициирует перекомпоновку формируемой таблицы по группам, каждая из которых имеет одинаковое значение в столбцах, включенных в перечень GROUP BY. Далее к этим группам применяются агрегирующие функции, указанные во фразе SELECT, что приводит к замене всех значений группы на единственное значение (сумма, количество и т.п.).

С помощью фразы HAVING (синтаксис которой почти не отличается от синтаксиса фразы WHERE):

HAVING [NOT] условие [[AND|OR][NOT] условие]...

можно исключить из результата группы, не удовлетворяющие заданным условиям:

значение { = | <> | < | <= | > | >= } { значение | (подзапрос) | SQL_функция}

В запросе SELECT, как уже было описано выше, могут использоваться SQL-функции, называемые еще *«агрегирующие функции»***.**

В стандарте языка определены следующие агрегирующие функции:
COUNT - число значений в столбце,

SUM - сумма значений в столбце,

AVG - среднее значение в столбце,

MAX - самое большое значение в столбце,

MIN - самое малое значение в столбце.

В качестве параметра в SQL-функцию передается имя столбца таблицы.

Для функций SUM и AVG рассматриваемый столбец должен содержать числовые значения.

Следует отметить, что здесь столбец - это столбец виртуальной таблицы, в которой могут содержаться данные не только из столбца базовой таблицы, но и данные, полученные путем функционального преобразования и (или) связывания символами арифметических операций значений из одного или нескольких столбцов. При этом выражение, определяющее столбец такой таблицы, может быть сколь угодно сложным, но не должно содержать SQL-функций (вложенность SQL-функций не допускается). Однако из SQL-функций можно составлять любые выражения.

Аргументу всех функций, кроме COUNT(*), может предшествовать ключевое слово DISTINCT (различный), указывающее, что избыточные дублирующие значения должны быть исключены перед тем, как будет применяться функция. Специальная же функция COUNT(*) служит для подсчета всех без исключения строк в таблице (включая дубликаты).

Если не используется фраза GROUP BY, то в перечень «эл-тов_выбора» (после слова SELECT) можно включать лишь SQL-функции или выражения, содержащие такие функции. Другими словами, нельзя иметь в списке столбцы, не являющихся аргументами SQL-функций.

Это связано с тем, что SQL-функция создает единственное значение из множества значений столбца-аргумента, а для "свободного" столбца должно быть выдано все множество его значений.

Фраза GROUP BY (группировать по) инициирует перекомпоновку указанной во FROM таблицы по группам, каждая из которых имеет одинаковые значения в столбце, указанном в GROUP BY.

Фраза HAVING играет такую же роль для групп, что и фраза WHERE для строк: она используется для исключения групп, точно так же, как WHERE используется для исключения строк. Эта фраза включается в предложение лишь при наличии фразы GROUP BY, а выражение в HAVING должно принимать единственное значение для группы.

Соединение

Даже при отсутствии средств одновременного доступа ко многим таблицам нежелателен проект, в котором информация о многих типах сущностей перемешана в одной таблице. SQL же обладает великолепным механизмом для одновременной или последовательной обработки данных из нескольких взаимосвязанных таблиц. В нем реализованы возможности "соединять" или "объединять" несколько таблиц и так называемые "вложенные подзапросы".

Суть соединения в том, что в списке элементов выбора указываются необходимые нам поля вне зависимости от того, в каких таблицах они располагаются. Сами таблицы, поля которых нам нужны, должны быть перечислены в разделе FROM. Кроме того, в секции условия WHERE мы должны указать условия, по которым записи из одной таблицы будут сопоставляться с записями другой таблицы (как правило, это условие о том, что значение поля внешнего ключа подчиненной таблицы должно быть равно ключевому полю главной).

Обрабатываться такой запрос будет следующим образом: СУБД последовательно формирует строки декартова произведения таблиц, перечисленных во фразе FROM, проверяет, удовлетворяют ли данные сформированной строки условиям фразы WHERE, и если удовлетворяют, то включает в ответ на запрос те ее поля, которые перечислены во фразе SELECT.

Следует подчеркнуть, что в SELECT и WHERE (во избежание двусмысленности) ссылки на все (*) или отдельные столбцы должны уточняться именем соответствующей таблицы.

Соединение таблицы с ее копией

В ряде приложений возникает необходимость одновременной обработки данных какой-либо таблицы и одной или нескольких ее копий, создаваемых на время выполнения запроса.

Например, при создании списков студентов (таблица Студенты) возможен повторный ввод данных о каком-либо студенте с присвоением ему второго номера зачетной книжки. Для выявления таких ошибок можно соединить таблицу Студенты с ее временной копией, установив в WHERE фразе равенство значений всех одноименных столбцов этих таблиц кроме столбцов с номером зачетной книжки (для последних надо установить условие неравенства значений).

Временную копию таблицы можно сформировать, указав с помощью ключевого слова AS имя псевдонима за именем таблицы во фразе FROM. Так, с помощью фразы

```
SELECT X.fio, Y.Nom_zach, Z.data_postupleniya
FROM Студенты AS X, Студенты AS Y, Студенты AS Z
```

будут сформированы три копии таблицы Студенты с именами X, Y и Z.

Вложенные подзапросы

Кроме механизма соединений в SQL есть механизм вложенных подзапросов, позволяющий объединить несколько простых запросов в едином предложении SELECT.

Вложенный подзапрос - это подзапрос, заключенный в круглые скобки и вложенный в WHERE (HAVING) фразу предложения SELECT или других предложений, использующих условие секции WHERE. Вложенный подзапрос может содержать в своем условии секции WHERE (HAVING) другой вложенный подзапрос и т.д. Нетрудно догадаться, что вложенный подзапрос создан для того, чтобы при отборе строк таблицы, сформированной основным запросом, можно было использовать данные из других таблиц (например, при отборе блюд для меню использовать данные о наличии продуктов в кладовой пансионата).

Существуют простые и коррелированные вложенные подзапросы. Они включаются в WHERE (HAVING) фразу с помощью условий IN, EXISTS или одного из условий сравнения (= | <> | < | <= | > | >=). Простые вложенные подзапросы обрабатываютя системой "снизу вверх". Первым обрабатывается вложенный подзапрос самого нижнего уровня. Множество значений, полученное в результате его выполнения, используется при реализации подзапроса более высокого уровня и т.д.

Запросы с коррелированными вложенными подзапросами обрабатываются системой в обратном порядке. Сначала выбирается первая строка рабочей таблицы, сформированной основным запросом, и из нее выбираются значения тех столбцов, которые используются во вложенном подзапросе (вложенных подзапросах). Если эти значения удовлетворяют условиям вложенного подзапроса, то выбранная строка включается в результат. Затем выбирается вторая строка и т.д., пока в результат не будут включены все строки, удовлетворяющие вложенному подзапросу (последовательности вложенных подзапросов). Коррелированный вложенный запрос называется еще иногда связанным подзапросом и отличается от простого тем, что в секции WHERE или HAVING этого вложенного запроса проверяется значение текущей записи внешнего запроса.

Таким образом, результат работы коррелированного запроса зависит от значений, определенных во внешнем подзапросе. Обработка коррелированного подзапроса, следовательно, должна повторяться для каждого значения извлекаемого из внешнего подзапроса, а не выполняться раз и навсегда.

Объединение (UNION)

В теории реляционных БД определена операция "Объединение", позволяющая получить отношение, состоящее из всех строк, входящих в одно или оба объединяемых отношения. Но при этом исходные отношения или их объединяемые проекции должны быть совместимыми по объединению. Для SQL это означает, что две таблицы (или запроса) можно объединять тогда и только тогда, когда:

– они имеют одинаковое число столбцов, например, m;

– для всех i (i = 1, 2, ..., m) i-й столбец первой таблицы и i-й столбец второй таблицы имеют в точности одинаковый тип данных.

Объединение запросов осуществляется при помощи оператора UNION и имеет следующий вид:

(SELECT ….) UNION (SELECT ….) UNION (SELECT ….) …

Причем, избыточные дубликаты всегда исключаются из результата UNION.

Модификация данных может выполняться с помощью предложений DELETE (удалить), INSERT (вставить) и UPDATE (обновить). Подобно предложению SELECT они могут оперировать как базовыми таблицами, так и представлениями. Однако по ряду причин не все представления являются обновляемыми.

Предложение DELETE имеет формат

DELETE
FROM базовая таблица | представление
[WHERE условие];

и позволяет удалить содержимое всех строк указанной таблицы (при отсутствии условия WHERE) или тех ее строк, которые выделяются условием WHERE.

Предложение INSERT имеет один из следующих форматов:

INSERT
INTO {базовая таблица | представление} [(столбец [,столбец] ...)]
VALUES ({константа | переменная} [,{константа | переменная}] ...);
или
INSERT
INTO {базовая таблица | представление} [(столбец [,столбец] ...)]
подзапрос;

В первом формате в таблицу вставляется строка со значениями полей, указанными в перечне фразы VALUES (значения), причем i-е значение соответствует i-му столбцу в списке столбцов (столбцы, не указанные в списке, заполняются NULL-значениями). Если в списке VALUES фразы указаны все столбцы модифицируемой таблицы и порядок их перечисления соответствует порядку столбцов в описании таблицы, то список столбцов в фразе INTO можно опустить. Однако не советуем этого делать, так как при изменении описания таблицы (перестановка столбцов или изменение их числа) придется переписывать и INSERT предложение.

Во втором формате сначала выполняется подзапрос, т.е. по предложению SELECT в памяти формируется рабочая таблица, а потом строки рабочей таблицы загружаются в модифицируемую таблицу. При этом i-й столбец рабочей таблицы (i-й элемент списка SELECT) соответствует i-му столбцу в списке столбцов модифицируемой таблицы. Здесь также при выполнении указанных выше условий может быть опущен список столбцов фразы INTO.

Предложение UPDATE также имеет два формата. Первый из них:

UPDATE (базовая таблица | представление}
SET столбец = значение [, столбец = значение] ...
[WHERE условие]
где значение - это
столбец | выражение | константа | переменная

и может включать столбцы лишь из обновляемой таблицы, т.е. значение одного из столбцов модифицируемой таблицы может заменяться на значение ее другого столбца или выражения, содержащего значения нескольких ее столбцов, включая изменяемый.

При отсутствии условия WHERE обновляются значения указанных столбцов во всех строках модифицируемой таблицы. Условие WHERE позволяет сократить число обновляемых строк, указывая условия их отбора.

Второй формат описывает предложение, позволяющее производить обновление значений модифицируемой таблицы по значениям столбцов из других таблиц. К сожалению, в ряде СУБД эти форматы отличаются друг от друга и от стандарта.

В значениях, находящихся в правых частях равенств фразы SET, следует уточнять имена используемых столбцов, предваряя их именем таблицы (псевдонима).

Базовые таблицы описываются в SQL с помощью предложения CREATE TABLE (создать таблицу), синтаксис которого имеет небольшие различия в различных СУБД. Однако все они поддерживают следующую минимальную форму:

```
CREATE TABLE базовая_таблица (столбец тип_данных [NOT NULL]
                              [,столбец тип_данных [NOT NULL]] ...);
```

где тип_данных должен принадлежать к одному из типов данных, поддерживаемых СУБД.

Например, описание таблицы Блюда может быть записано в виде:

```
CREATE TABLE Блюда
    (       БЛ      SMALLINT NOT NULL,
            Блюда   CHAR (70) NOT NULL,
            В       CHAR  (1),
            Основа  CHAR (10),
            Выход   FLOAT,
            Труд    SMALLINT );
```

В результате создается пустая базовая таблица Блюда, а в системный каталог помещается строка, описывающая эту таблицу. Отметим, что в профессиональных СУБД имя таблицы дополняется именем пользователя, который издал предложение CREATE TABLE. Если этот пользователь зарегистрирован в системе под именем Kirillov, то в каталоге будет зарегистрирована таблица Kirillov.Блюда и указанный пользователь может обращаться к ней по имени Kirillov.Блюда или по сокращенному имени Блюда.

Конструкция NOT NULL запрещает использование неопределенного значения, т.е. специального значения, которое вводится для представления "неизвестного значения" или "неприменимого значения".

Существующую базовую таблицу можно в любой момент уничтожить с помощью предложения DROP TABLE (уничтожить таблицу):

```
DROP TABLE базовая_таблица;
```

по которому удаляется описание таблицы, ее данные, связанные с ней представления и индексы, построенные для столбцов таблицы.

В SQL существует также предложение ALTER TABLE (изменить таблицу), которое позволяет добавить справа к таблице новый столбец, т.е. модифицировать описание таблицы.

В контексте баз данных термин безопасность означает защиту данных от несанкционированного раскрытия, изменения или уничтожения. SQL позволяет индивидуально защищать как целые таблицы, так и отдельные их поля. Для этого имеются две более или менее независимые возможности:

– механизм представлений, рассмотренный в предыдущей главе и используемый для скрытия засекреченных данных от пользователей, не обладающих правом доступа;

– подсистема санкционирования доступа, позволяющая предоставить указанным пользователям определенные привилегии на доступ к данным и дать им возможность

избирательно и динамически передавать часть выделенных привилегий другим пользователям, отменяя впоследствии эти привилегии, если потребуется.

Обычно при установке СУБД в нее вводится какой-то идентификатор, который должен далее рассматриваться как идентификатор наиболее привилегированного пользователя - системного администратора. Каждый, кто может войти в систему с этим идентификатором (и может выдержать тесты на достоверность), будет считаться системным администратором до выхода из системы. Системный администратор может создавать базы данных и имеет все привилегии на их использование. Эти привилегии или их часть могут предоставляться другим пользователям (пользователям с другими идентификаторами). В свою очередь, пользователи, получившие привилегии от системного администратора, могут передать их (или их часть) другим пользователям, которые могут их передать следующим и т.д.

Привилегии предоставляются с помощью предложения GRANT (предоставить), общий формат которого имеет вид:

GRANT привилегии ON объект TO пользователи;

В нем "привилегии" - список, состоящий из одной или нескольких привилегий, разделенных запятыми, либо фраза ALL PRIVILEGES (все привилегии); "объект" - имя и, если надо, тип объекта (база данных, таблица, представление, индекс и т.п.); "пользователи" - список, включающий один или более идентификаторов санкционирования, разделенных запятыми, либо специальное ключевое слово PUBLIC (общедоступный).

К таблицам (представлениям) относятся привилегии SELECT, DELETE, INSERT и UPDATE [(столбцы)], позволяющие соответственно считывать (выполнять любые операции, в которых используется SELECT), удалять, добавлять или изменять строки указанной таблицы (изменение можно ограничить конкретными столбцами). Например, предложение

GRANT SELECT, UPDATE (Труд) ON Блюда TO cook;

позволяет пользователю, который представился системе идентификатором cook, использовать информацию из таблицы Блюда, но изменять в ней он может только значения столбца Труд.

Если пользователь USER_1 предоставил какие-либо привилегии другому пользователю USER_2, то он может впоследствии отменить все или некоторые из этих привилегий. Отмена осуществляется с помощью предложения REVOKE (отменить), общий формат которого очень похож на формат предложения GRANT:

REVOKE привилегии ON объект FROM пользователи;

Например, можно отобрать у пользователя cook право изменения значений столбца Труд:

REVOKE UPDATE (Труд) ON Блюда FROM cook;

Транзакция или логическая единица работы, - это в общем случае последовательность ряда таких операций, которые преобразуют некоторое непротиворечивое состояние базы данных в другое непротиворечивое состояние, но не гарантируют сохранения непротиворечивости во все промежуточные моменты времени.

Никто кроме пользователя, генерирующего ту или иную последовательность SQL-предложений, не может знать о том, когда может возникнуть противоречивое состояние базы данных и после выполнения каких SQL-предложений оно исчезнет, т.е. база данных вновь станет актуальной. Поэтому в большинстве СУБД создается механизм обработки транзакций, при инициировании которого все изменения данных будут рассматриваются как предварительные до тех пор, пока пользователь (реже система) не выдаст предложения:

COMMIT (фиксировать)
- превращающее все предварительные обновления в окончательные ("зафиксированные");

ROLLBACK (откат)

- аннулирующее все предварительные обновления.

Таким образом, транзакцией можно назвать последовательность SQL-предложений, расположенных между "точками синхронизации", учреждаемых в начале выполнения программы и издании COMMIT или ROLLBACK и только в этих случаях. При этом следует иметь в виду, что возможен неявный COMMIT (существует режим AUTOCOMMIT, в котором система издает COMMIT после выполнения каждого SQL-предложения) и ROLLBACK (выполняемый при аварийном завершении программы).

Ясно теперь, что пользователь должен сам решать, включать ли механизм обработки транзакций и если включать, то где издавать COMMIT (ROLLEBACK), т.е. какие последовательности SQL-предложений являются транзакциями

Вопросы для самоконтроля:

-что такое запрос выборки данных SELECT. Основные секции запроса и их назначение;

-как использовать секцию WHERE. Используемые в ней логические операторы;

-приведите пример агрегирующих функций. Их значение и применение;

-как использовать ключевое слово DISTINCT в запросе;

-Оператор UNION и его назначение.

-какие вы знаете команды SQL для добавления, удаления и редактирования записей таблицы.

ЛАБОРАТОРНАЯ РАБОТА 3.1
ИСПОЛЬЗОВАНИЕ ЗАПРОСА ВЫБОРКИ ДАННЫХ ДЛЯ АНАЛИЗА ОТДЕЛЬНЫХ ТАБЛИЦ

Цель : научиться создавать запросы типа SELECT с помощью Конструктора запросов и вручную.

Материалы и оборудование: ПК

Индивидуальное задание: создать в СУБД Access базу данных из одной или нескольких таблиц. Заполнить каждую таблицу 15-20 записями и построить средствами СУБД запросы для анализа данных согласно варианту.

Требования к содержанию отчета

Отчет должен содержать: наименование работы, цель работы, постановку задачи, описание варианта задания, краткие теоретические сведения, описание хода выполнения лабораторной работы, результаты выполнения задания.

В качестве описания хода выполнения задания и результата работы должны быть приведены:

- исходные таблицы (таблица) по которым будут строиться запросы;
- распечатка текста каждого запроса и результирующей таблицы, полученной при выполнении запроса

Исходная таблица должна быть заполнена данными таким образом, чтобы результаты запроса наиболее ярко демонстрировали особенности использования тех секций запроса SELECT, которые оговорены в варианте.

Если требование варианта состоит из двух частей, то допускается выполнять оба требования варианта в рамках одного запроса или в виде отдельных.

Варианты заданий

1. Запрос на выбор всех данных по двум полям таблицы; запрос по произвольному количеству полей со сложным условием отбора записей;
2. Запрос на выбор всех НЕПОВТОРЯЮЩИХСЯ данных по одному полю таблицы; запрос с использованием условия сравнения строк с шаблоном (Like)
3. Запрос на выбор всех полей и записей таблицы, сгруппированных по значению одного поля, с использованием условия на группу (секции GROUP BY, HAVING) и с заголовками колонок, заданными в запросе.
4. Запрос на выбор всех неповторяющихся записей по одному полю таблицы с колонкой, образованной агрегирующей функцией SUM и озаглавленной в соответствии со смыслом.
5. Выбор нескольких (не всех) полей таблицы, отсортированных по УБЫВАНИЮ; выбор произвольного количества полей таблицы с добавлением поля, являющегося результатом арифметического выражения, в котором участвуют значения поля таблицы.
6. Запрос на выбор всех записей по одному полю таблицы с колонкой, образованной агрегирующей функцией SUM и озаглавленной в соответствии со смыслом; выбор записей с использованием условия диапазона (between)
7. Запрос на выбор всех записей по произвольному количеству полей таблицы с использование агрегирующей функции AVG и условием на отбор записей, заданном в секции WHERE.
8. Запрос на выбор двух полей таблицы с вычислением третьего поля по данным таблицы и с сортировкой по убыванию по первому полю, по возрастанию по второму и по убыванию по третьему.
9. Запрос на выборку данных таблицы с условием сравнения по шаблону LIKE; запрос с отбором по условию и сортировкой по убыванию одного из полей, а также добавлением поля, содержащего для всех записей константу, определенную при конструировании запроса.
10. Запрос с использованием агрегирующих функций MIN и MAX; запрос с использованием сложного условия с логическими операторами AND, OR и сортировкой.

Запросы строятся для БД, созданной в ходе выполнения предыдущих практических заданий.

ЛАБОРАТОРНАЯ РАБОТА 3.2
АНАЛИЗ ДАННЫХ СВЯЗАННЫХ ТАБЛИЦ
С ПОМОЩЬЮ СРЕДСТВ SQL

Цель: научиться извлекать данные из связанных таблиц средствами языка SQL.

Материалы и оборудование: ПК
Индивидуальное задание: создать в СУБД Access базу данных из нескольких связанных таблиц. Заполнить каждую таблицу **10-15** записями и построить средствами СУБД запросы для анализа данных согласно **варианту**.

Требования к содержанию отчета

Отчет должен содержать: наименование работы, цель работы, постановку задачи, описание варианта задания, краткие теоретические сведения, описание хода выполнения лабораторной работы, результаты выполнения задания.

В качестве описания хода выполнения лабораторной работы № 6 и результата работы должны быть приведены:

- исходные таблицы, по которым будут строиться запросы;
- распечатка текста каждого запроса и результирующей таблицы, полученной при выполнении запроса

Варианты заданий

Запросы строятся по базе данных, разработанной в ходе выполнения предыдущих практических заданий на базе вариантов, приведенных для лабораторной работы 1.1.

1 Запрос по двум таблицам с использованием внутреннего соединения; запрос по двум таблицам с помощью простого запроса SELECT с установлением связи с помощью условия WHERE, возвращающий **те же данные**, что и первый запрос (с внутренним соединением); связанный подзапрос по двум таблица;

2 Запрос по двум таблицам с левым внешним соединением; запрос **по тем же** таблицам с правым внешним соединением; вложенный подзапрос с использованием предиката SOME.

3 Сложный запрос по данным одной и той же таблицы (т.е. по таблице и ее псевдониму); простой запрос по нескольким таблицам по связи, установленной в секции WHERE; вложенный подзапрос с использованием предиката ALL.

4 Запрос с использованием UNION; запрос с использованием внутреннего соединения; вложенный подзапрос с использованием предиката ANY.

5 Вложенные подзапросы с использованием предиката EXISTS; запрос с использованием левого внешнего соединения; связанный подзапрос по двум таблицам.

ЛАБОРАТОРНАЯ РАБОТА 3.3
МОДИФИКАЦИЯ БАЗЫ И СТРУКТУР ТАБЛИЦ СРЕДСТВАМИ ЯЗЫКА SQL

Цель: научиться создавать и модифицировать базу данных с помощью запросов на обновление данных и модификацию таблиц языка SQL.

Материалы и оборудование: ПК

Индивидуальное задание: создать в СУБД Access базу данных из нескольких связанных таблиц. Заполнить каждую таблицу **10-15** записями и построить средствами СУБД запросы согласно **варианту**.

Требования к содержанию отчета

Отчет должен содержать: наименование работы, цель работы, постановку задачи, описание варианта задания, краткие теоретические сведения, описание хода выполнения лабораторной работы, результаты выполнения задания.

В качестве описания хода выполнения задания и результата работы должны быть приведены:

- исходные таблицы, по которым будут строиться запросы;
- распечатка текста каждого запроса и результирующей таблицы, полученной при выполнении запроса

Варианты задания

Для выполнения данного лабораторной работы следует использовать варианты к заданию 1.1. На основании схемы данных, разработанной при изучении темы 1, следует выполнить следующие этапы работы:

- создать с помощью SQL-запросов разработанную в первой лабораторной базу данных;
- составить SQL-запросы на добавление, удаление записей и редактирование значения полей;
- составить SQL-запросы на добавление и удаление столбцов таблиц и самих таблиц.

ЛИТЕРАТУРА

1. Атре Ш., Структурный подход к организации баз данных – М.: Финансы и статистика, 1983 – 317 с
2. Горев А. и др. Microsoft SQL Server 6.5 для профессионалов/ А.Горев, С. Макашарипов, Ю.Владимиров – С.-Пб.-М.-Харьков-Мн: Питер, 1998 – 464 с.
3. Грэй П., Логика, алгебра и базы данных – М.: Машиностроение, 1989 – 359 с.
4. Дейт, К., Руководство по РСУБД DB2 – М.: Финансы и статистика, 1988 – 319 с.
5. Джексон Г., Проектирование реляционных баз данных для использования с микроЭВМ – М.: Мир, 1991 – 252 с
6. Кочуров В.А. Современные базы данных - Минск: Дизайн Про, 1998 – 160 с.
7. Крамм Р., Системы управления базами данных dBase II и dBase III для ПК – М.: Финансы и статистика, 1988 – 383 с.
8. Мартин Дж., Организация баз данных в вычислительных системах. Изд. 2-е, доп. – М.: Мир, 1980 – 662 с.
9. Мейер Д., Теория реляционных баз данных – М.: Мир, 1987 – 608 с.
10. Методы и алгоритмы автоматизированного проектирования сложных систем управления – К.: Науковая думка, 1984 – 214 с.
11. Обгаров Л.А., Селетков С.Н., Автоматизированные банки данных – М.: Финансы и статистика, 1982 – 262 с.
12. Ремер С., Фратер Г. Microsoft Access: пер. С нем. – Киев: Торгово-издательское бюро BHV; М.: Фирма «Бином», 1994 – 384 с.
13. Сергиенко И.В. и др. Автоматизированные системы обработки данных – Киев.: Науковая думка, 1976 – 256 с.
14. Сигел Ч., Изучи сам Access 97. Русская версия: пер. с англ. – Мн.: Попурри, 1998 – 352 с
15. Тиори Т., Фрай Дж., Проектирование структур баз данных в 2х кн. – М.: Мир – 1985 – 287 с
16. Трофимов И.П. Системы обработки и хранения информации – М.: Высшая школа, 1989 – 190 с.
17. Ульман Дж. Базы данных на Паскале - Москва: Машиностроение, 1984 – 254 с
18. Ульман Дж., Основый систем баз данных – М.: Финансы и статистика, 1983 – 334 с.
19. Хаббард Дж.Ю. Автоматизированные проектирование баз данных – М.: Мир, 1984 – 203 с.
20. Четвериков В.Н. Базы и банки данных – М.:Высшая школа,1987 – 245 с

Учебное издание

ЛЕВАНЦОВ Виктор Николаевич

БАЗЫ И БАНКИ ДАННЫХ

Практическое пособие по лабораторным работам для студентов специальности I-53 01 02 – «АСОИ»

В авторской редакции

Подписано в печать __.__.20__ г. (__). Формат 60x84 1/16. Бумага писчая №1. Печать на ризографе. Гаринтура Times New Roman. Усл.печ.л.. Уч-изд.л.. Тираж __ экз.

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»,
246019, г.Гомель, ул.Советская, 104

Министерство образования республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

В.Н. ЛЕВАНЦОВ

БАЗЫ И БАНКИ ДАННЫХ

Практическое пособие
по выполнению лабораторных работ
часть 2

Гомель 2013

Введение

В настоящее время информационные системы, применяющие базы данных, представляют собой одну из важнейших областей современных компьютерных технологий. При построении корпоративных систем обработки данных формируется единое информационное пространство, работа в котором носит распределенный характер. Распределенная обработка данных реализуется в компьютерных сетях и требует определенной дисциплины взаимодействия. Общепринятым стандартом такого взаимодействия стала технология клиент-сервер, когда часть функций прикладной программы реализована на программе-клиенте, другая – на программе-сервере.

Серверы баз данных являются наиболее эффективным инструментом для создания приложений, оперирующими большими объемами информации и являются важнейшим звеном в их построении по схеме клиент-сервер. При этом эффективно реализуется интегрированность базы данных, то есть возможность одновременного доступа к данным из нескольких приложений. Создание клиент-серверного приложения, работающего с базой данных, требует прохождения следующих этапов:

1 – разработка структуры реляционной базы данных;

2 – администрирование базы данных на стороне сервера;

3 – программирование на стороне сервера;

4 – программирование на стороне клиента.

В данном методическом руководстве рассматриваются все этапы построения клиент-серверного приложения на примере создания системы делопроизводства деканата вуза. В настоящий момент, одной из самых популярных серверных платформ, является Microsoft SQL Server, на котором и построена рассматриваемая база данных. Клиентское приложение строится на основе СУБД Microsoft Visual Foxpro. Foxpro является специализированным языком программирования, который ориентирован на работу с базами данных. Эта программная среда позволяет создавать эффективные приложения, работающие с локальными, сетевыми и удаленными базами данных на разных серверных платформах, в том числе и на Microsoft SQL Server.

В предлагаемой работе ввод данных и их программирование осуществляется и на стороне клиента, и на стороне сервера. Используются все средства, входящие в состав Visual Foxpro для формирования клиент-серверного приложения и взаимодействия с сервером баз данных на основе Microsoft SQL Server. Взаимодействие клиентской и серверной части разрабатываемой базы данных построено таким образом, чтобы продемонстрировать основные возможности Microsoft SQL Server по управлению входящими потоками информации:

- с помощью триггеров, программируемых на стороне сервера, повышается скорость обработки входящих данных и снижается трафик в сети организации;

- разработка транзакций на сервере позволяет производить откат некорректно выполненных операций;

- с помощью средств мониторинга и сопровождения SQL Server решается ряд задач по администрированию сервера.

Таким образом целью данной работы является рассмотрение основных возможностей Microsoft SQL Server по управлению базой данных и решение основных задач по администрированию сервера, построение клиентских приложений для серверных платформ и организация взаимодействия сервера с клиентским программным обеспечением.

Данное методическое руководство разработано для «специальности» «АСОИ» и предназначено для изучения курса «Б и БД».

Лабораторная работа №1 Разработка базы данных на Microsoft SQL Server

Разработка клиент-серверной информационной системы начинается с разработки базы данных на стороне сервера и настройки серверной платформы. Здесь можно выделить следующие задачи:

1 – создание базы данных и установка ее свойств;

2 - разработка таблиц;

3 – установление отношений между таблицами и обеспечение целостности данных;

4 – программирование на стороне сервера, написание триггеров и транзакций;

5 - ввод первоначальных данных.

1.1. Создание базы данных и установка ее свойств.

Создайте новую базу данных.

Откройте Enterprise Manager, найдите в консоли папку Databases на сервере, который вы используете. SQL Server отобразит список баз данных (рис.1.1).

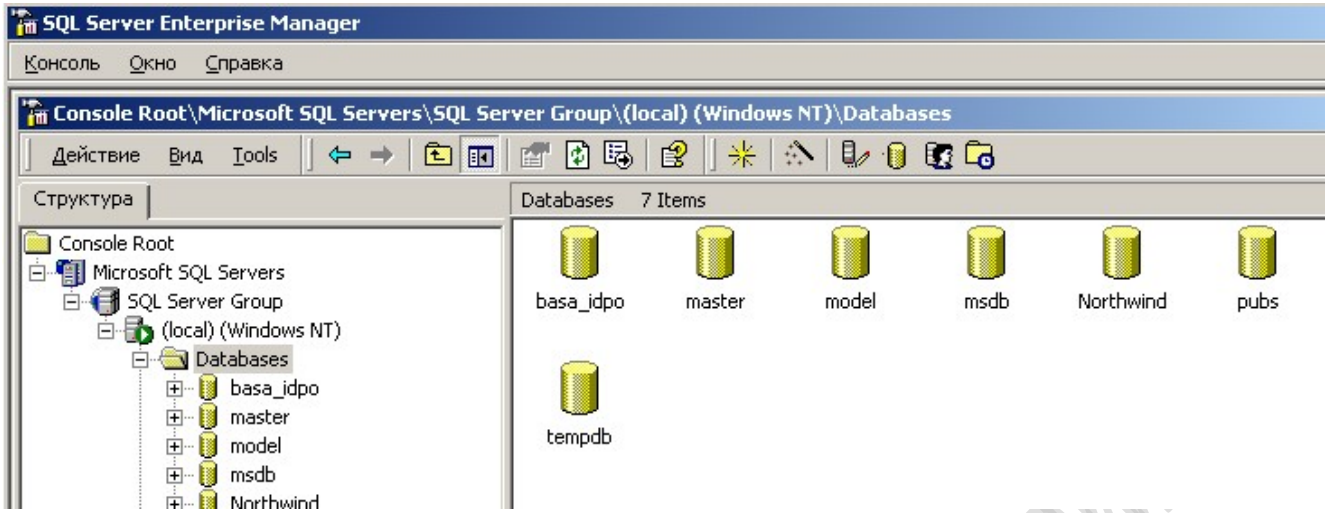


Рис.1.1 Создание базы данных

Выберите инструмент «New Database» и введите имя создаваемой базы данных.

Размещение файлов базы данных.

В процессе эксплуатации базы данных неоднократно возникают задачи переноса базы данных с одного носителя на другой и управления производительностью работы сервера. При создании новой базы данных формируются два файла ее сопровождения:

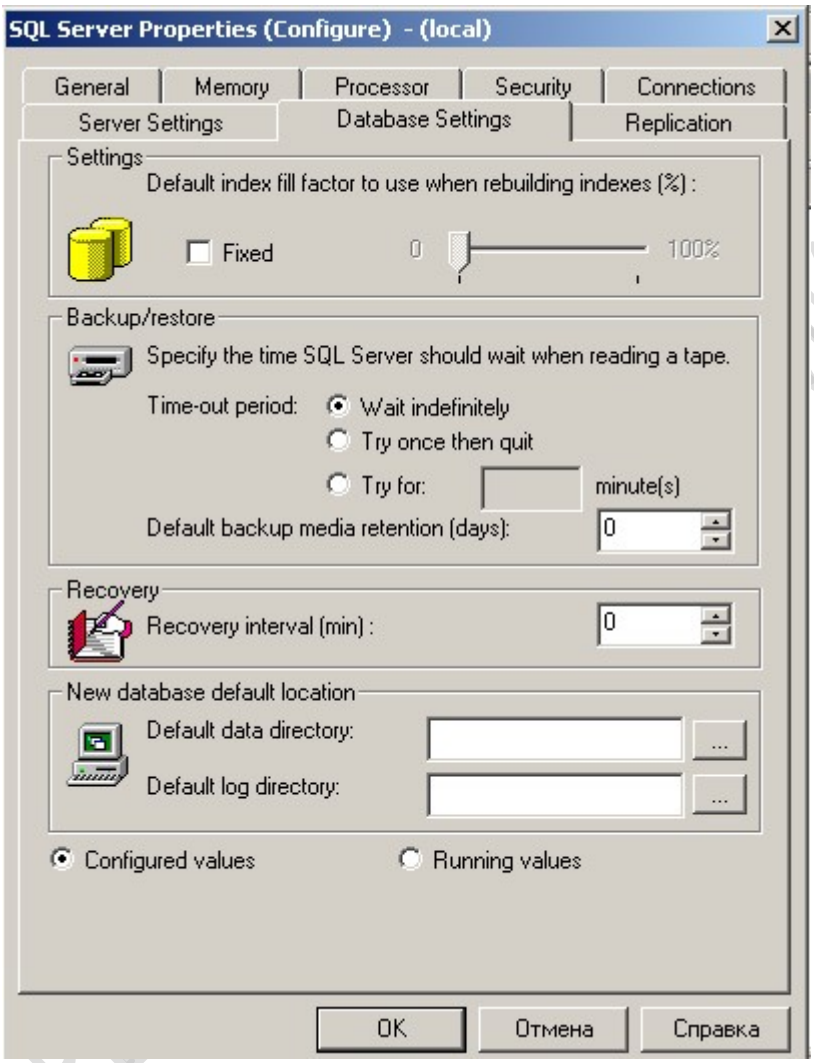
*.MDF (Master Data File)– файл данных, предназначен для хранения информации, находящейся в таблицах базы данных. Кроме того, в этом файле так же размещены процедуры, ограничения, триггеры, индексы и другая информация;

*.LDF – файл журнала транзакций, хранит информацию о ходе выполнения транзакций. В нем размещается информация о состоянии данных перед началом транзакции, о выполняемых изменениях, блокированных ресурсах и другая сопутствующая информация.

Любая база данных должна содержать как минимум один файл данных и один файл журнала транзакций. При необходимости администратор может добавлять в БД новые файлы данных или файлы журнала транзакций. Если компьютер, на котором установлен SQL Server имеет несколько физических дисков, то для повышения производительности Microsoft настоятельно рекомендует для каждой базы данных создать как минимум один файл на каждом физическом диске. Кроме того, по возможности следует располагать файлы данных и журнала транзакций на отдельных физических дисках. Это повышает производительность работы всего сервера баз данных.

При первой установке SQL Server для новых баз данных по умолчанию принимается место размещения: \диск установки SQL Server\Program Files\Microsoft SQL Server*MSSQL\data*. Эта настройка по умолчанию предоставляется мастеру создания базы данных Create Database Wizard. Чтобы изменить эту используемую по умолчанию установку, вы можете задать новое место размещения на вкладке Database Settings (Параметры базы данных) в диалоговом окне SQL Server Properties (Свойства SQL Server). Для этого:

1. Щелкните правой кнопкой мыши на сервере в дереве консоли Console Tree, выберите Properties (Свойства), а за тем откройте вкладку Database Settings (Параметры базы данных).



2. Перейдите к разделу Default data directoty и щелкните на кнопке Browse (Обзор), чтобы изменить местоположение файла базы данных. Мастер отобразит диалоговое окно, запрашивающее новое место размещения. Укажите нужную вам папку для размещения файла базы данных.

3. Перейдите к разделу Default log directoty и щелкните на кнопке Browse (Обзор), чтобы изменить местоположение файла журнала транзакций. Мастер отобразит диалоговое окно, запрашивающее новое место размещения. Укажите нужную вам папку для размещения файла базы данных.

Установка свойств базы данных.

Когда вы создаете базу данных с помощью мастера Create Database Wizard, вы указываете определенные характеристики, или свойства, базы данных, такие как имя базы данных и место размещения. После создания базы данных вы можете поменять эти свойства, изменив соответствующие параметры в диалоговом окне Properties (Свойства).

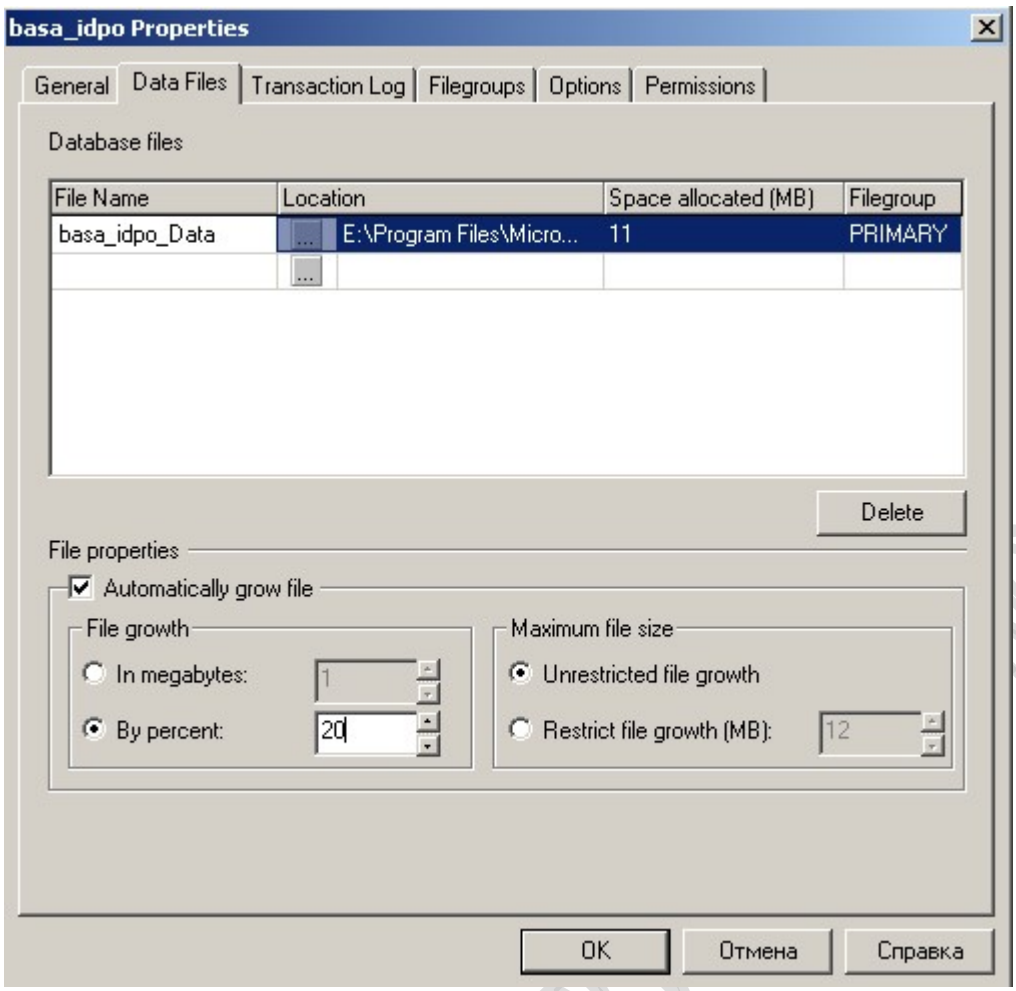
Например, увеличение размера физического файла — это довольно серьезная операция, выполнение которой может привести к увеличению времени отклика сервера. Если вы обнаружите, что SQL Server приходится слишком часто увеличивать размер файла. Вам следует учитывать возможность изменения процента увеличения размера файла в диалоговом окне Properties (Свойства), чтобы сервер смог увеличивать размер файла более чем на 10% (установка по умолчанию).

Изменение процента увеличения размера файла.

1. Выберите свою базу данных в дереве консоли Console Tree.

2. Нажмите кнопку Properties (Свойства) в панели инструментов. SQL Server отобразит диалоговое окно Properties (Свойства) для базы данных.

3. Откройте вкладку Data Files (Файлы данных). SQL Server отобразит свойства файлов данных базы данных.
4. Установите процент увеличения 20%.



5. Нажмите OK. SQL Server установит новое свойство и закроет диалоговое окно Properties (Свойства).

1.2. Создание таблиц в SQL Server.

Разработайте следующие таблицы, используя Enterprise Manager (рис.1):

студенты

	Column Name	Data Type	Length	Allow Nulls
🔑	id_студент	int	4	
	номер_дела	varchar	20	✓
	фамилия	varchar	20	✓
	имя	varchar	20	✓
	отчество	varchar	20	✓
	FK_специальность	int	4	✓
	FK_курс	int	4	✓
	FK_поток	int	4	✓
	место_работы	varchar	100	✓
	год_рождения	smalldatetime	4	✓
	соц_положение	varchar	15	✓
	адрес	varchar	80	✓
	образование	varchar	100	✓
	приказ_зачисления	varchar	30	✓
	дата_значения	smalldatetime	4	✓
	основание_зачисления	varchar	60	✓
	приказ_отчисления	varchar	30	✓
	дата_отчисления	smalldatetime	4	✓
	причина_отчисления	varchar	60	✓
	приказ_восстановления	varchar	30	✓
	дата_восстановления	smalldatetime	4	✓
	приказ_академа	varchar	30	✓
	дата_академа	smalldatetime	4	✓
	причина_академа	varchar	60	✓
	приказ_доплома	varchar	30	✓
	дата_диплома	smalldatetime	4	✓
	тема_диплома	varchar	250	✓
	оценка_диплома	int	4	✓
	квалификация	varchar	40	✓
	приказ_2_курс	varchar	10	✓
	дата_2_курс	smalldatetime	4	✓
	приказ_3_курс	varchar	10	✓
	дата_3_курс	smalldatetime	4	✓

успеваемость_студ

	Column Name	Data Type	Length	Allow Nulls
🔑	id_усп	int	4	
	ведомость	varchar	50	✓
	FK_спец	int	4	✓
	FK_курс	int	4	✓
	FK_поток	int	4	✓
	FK_предмет	int	4	✓
	FK_препод	int	4	✓
	семестр	int	4	✓
	часы	int	4	✓
	дата	smalldatetime	4	✓

курсовая

	Column Name	Data Type	Length	Allow Nulls
🔑	id_курс	int	4	
	курсовая	varchar	15	✓

дисцип_препод

	Column Name	Data Type	Length	Allow Nulls
🔑	id_дисцип_препод	int	4	
	FK_преподаватель	int	4	✓
	FK_предмет	int	4	✓
	FK_специальность	int	4	✓

учебный_план

	Column Name	Data Type	Length	Allow Nulls
🔑	id_уч_план	int	4	
	шифр_спец	varchar	50	✓
	FK_специальность	int	4	✓
	FK_дисциплина	int	4	✓
	FK_курсовая	int	4	✓
	FK_контрольная	int	4	✓
	FK_форма_контр	int	4	✓
	шифр_предмет	varchar	20	✓
	итого	int	4	✓
	лекции	int	4	✓
	практика	int	4	✓
	сам_работа	int	4	✓
	семестр	varchar	10	✓
	номер	int	4	✓

преподаватели

	Column Name	Data Type	Length	Allow Nulls
🔑	id_преподаватель	int	4	
	фамилия	varchar	50	✓
	имя	varchar	50	✓
	отчество	varchar	50	✓
	e_mail	varchar	50	✓
	место_работы	varchar	50	✓
	уч_степень	varchar	50	✓
	звание	varchar	50	✓
	фото	varchar	50	✓

оценка_студ

	Column Name	Data Type	Length	Allow Nulls
🔑	id_оценка	int	4	
	FK_усп	int	4	✓
	FK_студ	int	4	✓
	FK_оценка	int	4	✓
	FK_контр	int	4	✓
	FK_курс	int	4	✓
	FK_форма_контр	int	4	✓

специальности

	Column Name	Data Type	Length	Allow Nulls
🔑	id_спец	int	4	
	специальность	varchar	50	✓

предметы

	Column Name	Data Type	Length	Allow Nulls
🔑	id_предмет	int	4	
	предмет	varchar	150	✓

курс

	Column Name	Data Type	Length	Allow Nulls
🔑	id_курс	int	4	
	курс	varchar	10	✓

оценка

	Column Name	Data Type	Length	Allow Nulls
🔑	id_оценка	int	4	
	оценка	varchar	20	✓

контрольная

	Column Name	Data Type	Length	Allow Nulls
🔑	id_кр	int	4	
	контрольная	varchar	15	✓

поток

	Column Name	Data Type	Length	Allow Nulls
🔑	id_поток	int	4	
	поток	varchar	10	✓

ф_контроля

	Column Name	Data Type	Length	Allow Nulls
🔑	id_ф_контроля	int	4	
	контроль	varchar	50	✓

Рис.1.1. Таблицы базы данных

1.3. Создание связей между таблицами (рис.1.2)

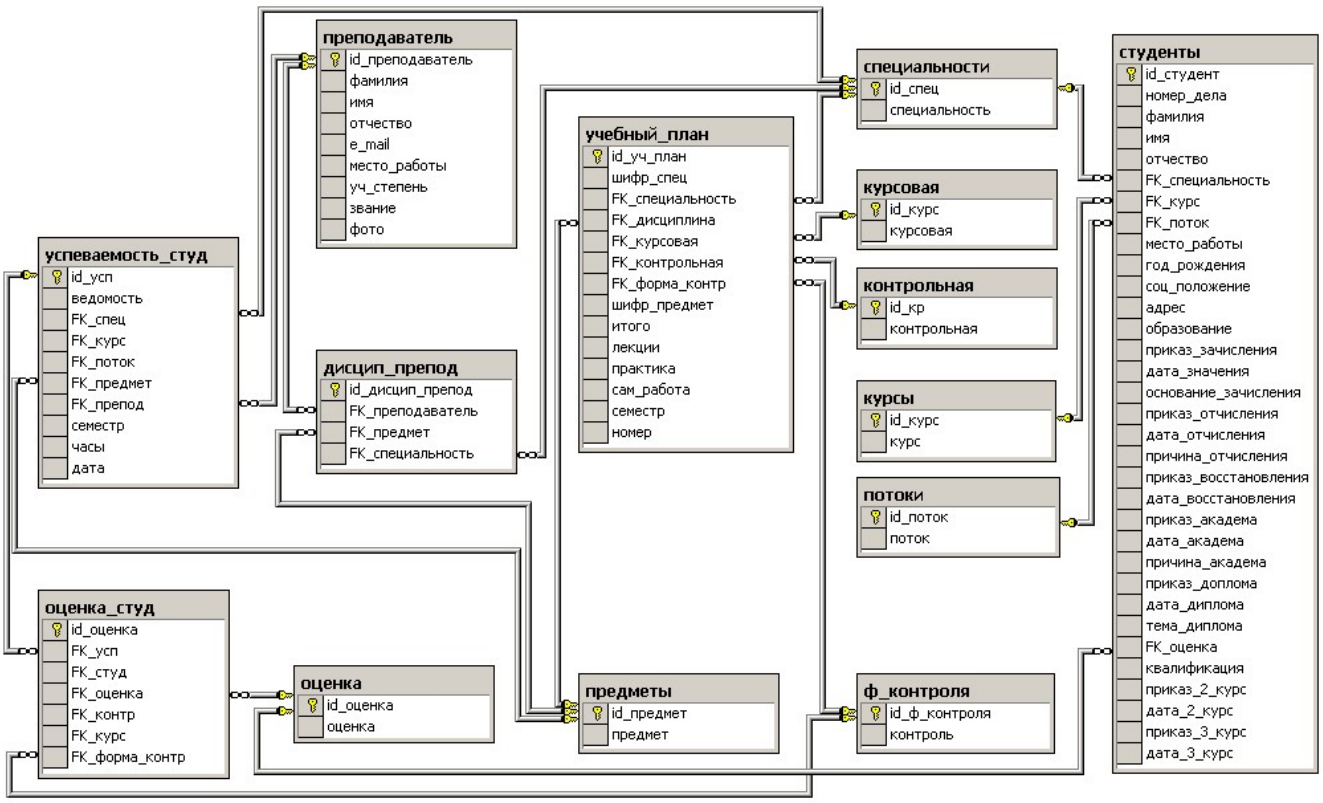


Рис.1.2. Схема базы данных

Таблицы связаны по следующим полям:

1- таблица «преподаватель», поле «id_преподаватель» – таблица «успеваемость_студ», поле «FK_препод»;

2- таблица «преподаватель», поле «id_преподаватель» – таблица «дисцип_препод», поле «FK_преподаватель»;

3- таблица «успеваемость_студ», поле «id_усп» – таблица «оценка_студ», поле «FK_усп»;

4- таблица «предметы», поле «id_ предмет» – таблица «учебный_план», поле «FK_дисциплина»;

5- таблица «предметы», поле «id_ предмет» – таблица «дисцип_препод», поле «FK_предмет»;

6- таблица «специальности», поле «id_спец» – таблица «учебный_план», поле «FK_специальность»;

7- таблица «специальности», поле «id_ спец» – таблица «дисцип_препод», поле «FK_специальность»;

8- таблица «специальности», поле «id_ спец» – таблица «студенты», поле «FK_специальность»;

9- таблица «потоки», поле «id_ поток» – таблица «студенты», поле «FK_поток»;

10- таблица «курсы», поле «id_ курс» – таблица «студенты», поле «FK_курс»;

11- таблица «оценка», поле «id_ оценка» – таблица «студенты», поле «FK_оценка»;

12 - таблица «предметы», поле «id_ предмет» – таблица «успеваемость_студ», поле «FK_предмет»;

13 - таблица «специальности», поле «id_ спец» – таблица «успеваемость_студ», поле «FK_спец»;

14- таблица «оценка», поле «id_ оценка» – таблица «оценка_студ», поле «FK_оценка».

Лабораторная работа №2 Создание запросов и модификация таблиц базы данных.

Цель работы: Используя данные базы данных, подготовленной в предыдущей лабораторной работе, подготовить и реализовать серию запросов, связанных с выборкой информации и модификацией данных таблиц.

Содержание работы и методические указания к ее выполнению

1. Изучить набор команд языка SQL, связанный с созданием запросов, добавлением, модификацией и удалением строк таблицы:

select - осуществление запроса по выборке информации из таблиц базы данных;

insert - добавление одной или нескольких строк в таблицу;

delete - удаление одной или нескольких строк из таблицы;

update - модификация одной или нескольких строк таблицы;

union - объединение запросов в один запрос.

2. Изучить состав, правила и порядок использования ключевых фраз оператора select:

select - описание состава данных, которые следует выбрать по запросу (обязательная фраза);

from - описание таблиц, из которых следует выбирать данные (обязательная фраза);

where - описание условий поиска и соединения данных при запросе;

group by - создание одной строки результата для каждой группы (группой называется множество строк, имеющих одинаковые значения в указанных столбцах);

having - наложение одного или более условий на группу;

order by - сортировка результата выполнения запроса по одному или нескольким столбцам;

into outfile - создание файла, в который будет осуществлен вывод результатов соответствующего запроса.

Порядок следования фраз в команде select должен соответствовать приведенной выше последовательности. Для лучшего понимания механизма функционирования выполните следующие упражнения:

I. Простые запросы на языке SQL

Запрос на языке SQL формируется с использованием оператора Select. Оператор Select используется

- для выборки данных из базы данных;
- для получения новых строк в составе оператора Insert;

- для обновления информации в составе оператора Update.

В общем случае оператор Select содержит следующие семь спецификаторов, расположенных в операторе в следующем порядке:

- спецификатор Select;
- спецификатор From;
- спецификатор Where;
- спецификатор Group by;
- спецификатор Having;
- спецификатор Order by;

Обязательными являются только спецификаторы Select и From. Эти два спецификатора составляют основу каждого запроса к базе данных, поскольку они определяют таблицы, из которых выбираются данные, и столбцы, которые требуется выбрать.

Спецификатор Where добавляется для выборки определенных строк или указания условия соединения. Спецификатор Order by добавляется для изменения порядка получаемых данных. Спецификатор Into temp добавляется для сохранения этих результатов в виде таблицы с целью выполнения последующих запросов. Два дополнительных спецификатора оператора Select - Group by (спецификатор группирования) и Having (спецификатор условия выборки группы) - позволяют выполнять более сложные выборки данных.

У п р а ж н е н и я

1. Выбор всех строк и столбцов таблицы.

Пример.

Выдать полную информацию о поставщиках.

Select * from S

Результат: таблица S в полном объеме.

Подготовьте запрос и проверьте полученный результат.

2. Измение порядка следования столбцов.

Пример.

Выдать таблицу S в следующем порядке: фамилия, город, рейтинг, номер_поставщика.

Select фамилия, город, рейтинг, номер_поставщика from S

Результат: таблица S в требуемом порядке.

Подготовьте запрос и проверьте полученный результат.

3. Выбор заданных столбцов.

Пример.

Выдать номера всех поставляемых деталей.

Select номер_детали from SPJ

Результат: столбец номер_детали таблицы SPJ

Подготовьте запрос и проверьте полученный результат.

4. Выбор без повторения.

Пример.

Выдать номера всех поставляемых деталей, исключая дублирование.

Select distinct номер_детали from SPJ

Результат:	**номер_детали**
	P1
	P2
	P3
	P4
	P5
	P6

Подготовьте запрос и проверьте полученный результат.

5. Использование в запросах констант и выражений.

Пример.

Select номер_детали, "вес в граммах", вес*454 from P

Результат:	P1 вес в граммах=5448

	P6 вес в граммах=8226

Подготовьте запрос и проверьте полученный результат.

6.Ограничение в выборке.

Пример.

Выдать номера всех поставщиков, находящихся в Париже с рейтингом > 20.

Select номер_поставщика from S where город="Париж" and рейтинг>20

Результат:	номер_поставщика
	S3

Подготовьте запрос и проверьте полученный результат.

7. Выборка с упорядочиванием.

Пример.

Выдать номера поставщиков, находящихся в Париже в порядке убывания рейтинга.

Select номер_поставщика, рейтинг from S where город="Париж" order by рейтинг desc

Результат:	номер_поставщика	рейтинг
	S3	30
	S2	10

Подготовьте запрос и проверьте полученный результат.

8. Упорядочивание по нескольким столбцам.

Пример.

Выдать список поставщиков, упорядоченных по городу, в пределах города - по рейтингу.

Select * from S order by 4, 3

Результат:	Номер_поставщика	Фамилия	Рейтинг	Город
	S5	Адамс	30	Атенс
	S1	Смит	20	Лондон
	S4	Кларк	20	Лондон
	S2	Джонс	10	Париж
	S3	Блейк	30	Париж

Подготовьте запрос и проверьте полученный результат.

9. Фраза in (not in).

Пример.

Выдать детали, вес которых равен 12, 16 или 17.

Select номер_детали, название, вес from P where вес in (12, 16, 17)

Результат:	номер_детали	Название	вес
	P1	Гайка	12
	P2	Болт	17
	P3	Винт	17
	P5	Кулачок	12

Подготовьте запрос и проверьте полученный результат.

12. Выбор по шаблону.

Для запросов с поиском по шаблону, основанных на поиске подстрок в полях типа CHARACTER, используются ключевое слова LIKE.

Включение в выражение ключевого слова NOT порождает условие с обратным смыслом. Ключевое слово LIKE соответствует стандарту ANSI.

СИМВОЛ	ЗНАЧЕНИЕ
LIKE	
%	Заменяет последовательность символов
-	Заменяет любой одиночный символ
\	Отменяет специальное назначение следующего за ним символа

Примеры.

а) Выбрать список деталей, начинающихся с буквы "Б"[1]

Select номер_детали, название, вес from P where название like "Б%"

Результат:	номер_детали	название	вес
	P5	Болт	12
	P6	Блюм	19

II. Использование функций

1. Агрегатные функции.

Примеры.

а) Выдать общее количество поставщиков.

Select count (*) from S

Результат: 5

Подготовьте запрос и проверьте полученный результат.

б) Выдать общее количество поставщиков, поставляющих в настоящее время детали.

Select count (distinct номер_поставщика) from SPJ

Результат: 4

Подготовьте запрос и проверьте полученный результат.

в) Выдать количество поставок для детали P2.

Select count (*) from SPJ where номер_детали='P2'

Результат: 5

Подготовьте запрос и проверьте полученный результат.

г) Выдать общее количество поставляемых деталей 'P2'.

Select sum (количество) from SPJ where номер_детали='P2'

[1] Примечание. Корректно работает только при задании кодировки по умолчанию. Задается в разделе MYSQLD default_character_set=win1251

Результат: 1000

Подготовьте запрос и проверьте полученный результат.

д) Выдать средний, минимальный и максимальный объем поставок для поставщика S1 с соответствующим заголовком.

Select avg(количество) average, min(количество) minimum,

max(количество) maximum from SPJ where номер_поставщика='S1'

Результат:	average	minimum	maximum
	216.6	100	400

Подготовьте запрос и проверьте полученный результат.

2. Ниже привиден перечень всех функций, используемых в операторе Select

Функции

select_expression может содержать следующие функции и операторы:

+ - * /	Арифметические действия.
%	Остаток от деления (как в C)
| &	Битовые функции (используется 48 бит).
- C	Мена знака числа.
( )	Скобки.
BETWEEN(A, B, C)	(A >= B) AND (A <= C).
BIT_COUNT()	Количество бит.
ELT(N, a, b, c, d)	Возвращает a, если N == 1, b, если N == 2 и т. д. a, b, c, d строки. **ПРИМЕР:** ELT(3, "First", "Second", "Third", "Fourth") вернет "Third".
FIELD(Z, a, b, c)	Возвращает a, если Z == a, b, если Z == b и т. д. a, b, c, d строки. **ПРИМЕР:** FIELD("Second", "First", "Second", "Third", "Fourth") вернет "Second".

IF(A, B, C)	Если A истина (!= 0 and != NULL), то вернет B, иначе вернет C.
IFNULL(A, B)	Если A не null, вернет A, иначе вернет B.
ISNULL(A)	Вернет 1, если A == NULL, иначе вернет 0. Эквивалент ('A == NULL').
NOT !	NOT, вернет TRUE (1) или FALSE (0).
OR, AND	Вернет TRUE (1) или FALSE (0).
SIGN()	Вернет -1, 0 или 1 (знак аргумента).
SUM()	Сумма столбца.
= <> <= < >= >	Вернет TRUE (1) или FALSE (0).
expr LIKE expr	Вернет TRUE (1) или FALSE (0).
expr NOT LIKE expr	Вернет TRUE (1) или FALSE (0).
expr REGEXP expr	Проверяет строку на соответствие регулярному выражению expr.

select_expression может также содержать один или большее количество следующих математических функций.

ABS()	Абсолютное значение (модуль числа).
CEILING()	()
EXP()	Экспонента.
FORMAT(nr, NUM)	Форматирует число в формат '#, ###, ###.##' с NUM десятичных цифр.
LOG()	Логарифм.
LOG10()	Логарифм по основанию 10.
MIN(), MAX()	Минимум или максимум соответственно. Должна иметь при вызове два или более аргументов, иначе рассматривается как групповая функция.
MOD()	Остаток от деления (аналог %).
POW()	Степень.
ROUND()	Округление до ближайшего целого числа.

RAND([integer_expr])	Случайное число типа float, 0 <= x <= 1.0, используется integer_expr как значение для запуска генератора.
SQRT()	Квадратный корень.

select_expression может также содержать одну или больше следующих строковых функций.

CONCAT()	Объединение строк.
INTERVAL(A, a, b, c, d)	Возвращает 1, если A == a, 2, если A == b... Если совпадений нет, вернет 0. A, a, b, c, d... строки.
INSERT(org, strt, len, new)	Заменяет подстроку org[strt...len(gth)] на new. Первая позиция строки=1.
LCASE(A)	Приводит A к нижнему регистру.
LEFT()	Возвращает строку символов, отсчитывая слева.
LENGTH()	Длина строки.
LOCATE(A, B)	Позиция подстроки B в строке A.
LOCATE(A, B, C)	Позиция подстроки B в строке A, начиная с позиции C.
LTRIM(str)	Удаляет все начальные пробелы из строки str.
REPLACE(A, B, C)	Заменяет все подстроки B в строке A на подстроку C.
RIGHT()	Get string counting from right.
RTRIM(str)	Удаляет хвостовые пробелы из строки str.
STRCMP()	Возвращает 0, если строки одинаковые.
SUBSTRING(A, B, C)	Возвращает подстроку из A, с позиции B до позиции C.
UCASE(A)	Переводит A в верхний регистр.

Еще несколько просто полезных функций, которые тоже можно применить в select_expression.

CURDATE()	Текущая дата.
DATABASE()	Имя текущей базы данных из которой выполняется выбор.
FROM_DAYS()	Меняет день на DATE.
NOW()	Текущее время в форматах YYYYMMDDHHMMSS или "YYYY-MM-DD HH:MM:SS". Формат зависит от того в

	каком контексте используется NOW(): числовом или строковом.
PASSWORD()	Шифрует строку.
PERIOD_ADD(P:N)	Добавить N месяцев к периоду P (в формате YYMM).
PERIOD_DIFF(A, B)	Возвращает месяцы между A и B. Обратите внимание, что PERIOD_DIFF работает только с датами в форме YYMM или YYYMM.
TO_DAYS()	Меняет DATE (YYMMDD) на номер дня.
UNIX_TIMESTAMP([date])	Возвращает метку времени unix, если вызвана без date (секунды, начиная с GMT 1970.01.01 00:00:00). При вызове со столбцом TIMESTAMP вернет TIMESTAMP. date может быть также строкой DATE, DATETIME или числом в формате YYMMDD (или YYYMMDD).
USER()	Возвращает логин текущего пользователя.
WEEKDAY()	Возвращает день недели (0 = понедельник, 1 = вторник, ...).

Групповые функции в операторе ***select*:**

Следующие функции могут быть использованы в предложении GROUP:

AVG()	Среднее для группы GROUP.
SUM()	Сумма элементов GROUP.
COUNT()	Число элементов в GROUP.
MIN()	Минимальный элемент в GROUP.
MAX()	Максимальный элемент в GROUP.

Задание:

1. Подготовить 3 запроса с использованием различных функций работа с полем дата, со строковыми данными (в том числе групповых).

2. Подготовить и выполнить средствами СУБД MySQL 4 запроса по выборке информации из таблиц базы данных с использованием агрегатных функций..

4. Подготовить и выполнить средствами СУБД MySQL 2 запроса по модификации информации (вставка, удаление, замещение) из таблиц базы данных для решения нижеприведенных задач. При этом в тех заданиях, где речь идет о создании таблиц, предполагается формировании постоянной таблицы базы данных.

Лабораторная работа №3. Представления, курсоры, хранимые процедуры, функции, триггеры

Цель работы: познакомиться с возможностями MySQL по работе с хранимыми процедурами, функциями, триггерами, представлениями.

Представления

Представления (views) можно сравнить с временными таблицами, наполненными динамически формируемым содержимым.. В настоящей реализации есть две возможности создания представлений: с использованием алгоритма временных таблиц MySQL и с созданием самостоятельной таблицы. Нас интересует именно второй способ (первый был реализован, скорее всего, исходя из соображений совместимости и унификации). Такие представления позволяют значительно снизить объём кода, в котором часто повторялись простые объединения таблиц. К ним (после создания) применимы любые запросы, возвращающие результат в виде набора строк. То есть команды SELECT, UPDATE, DELETE, можно применять так же, как и к реальным таблицам. Важно и то, что посредством представлений можно более гибко распоряжаться правами пользователей базы данных, так как в этом случае есть возможность предоставлять доступ на уровне отдельных записей различных таблиц.

Создание представлений

Для создания представлений используется команда CREATE VIEW

Синтаксис команды CREATE VIEW

```
CREATE
  [OR REPLACE]
  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
```

[DEFINER = { *user* | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW *view_name* [(*column_list*)]
AS *select_statement*
[WITH [CASCADED | LOCAL] CHECK OPTION]

Пример создания и работы простейшего представления:

Create View v as Select column 1 from T

Insert into v Values (1)

Select * from v

Результат

```
+---------+
| column1 |
+---------+
| 1       |
+---------+
1 row in set (0.00 sec)
```

Представление может быть создано на основе различных параметров предложения SELECT, при этом можно ссылаться на другие таблицы и представления. Конструкция может использовать оператор UNION и другие подзапросы.

Синтаксис команды ALTER VIEW

Для внесения изменений в представление используется команда ALTER VIEW

ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { *user* | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW *view_name* [(*column_list*)]
AS *select_statement*
[WITH [CASCADED | LOCAL] CHECK OPTION]

Синтаксис команды DROP VIEW

Для удаления представления используется команда DROP VIEW

VIEW [IF EXISTS]
view_name [, *view_name*] ...
[RESTRICT | CASCADE]

ПРИМЕР

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
```

Хранимые процедуры и функции

В СУБД MySQL появилась возможность создания и хранения функций и процедур. Объявление и работа с процедурами и функциями отличаются в следующем:

- в заголовке функции помимо описания формальных параметров обязательно указывается тип возвращаемого ею результата;
- для возврата функцией значения в точку вызова среди ее операторов должен быть хотя бы один, в котором имени функции или переменной Result присваивается значение результата;
- вызов процедуры выполняется отдельным оператором;
- вызов функции может выполняться там, где допускается ставить выражение, в частности, в правой части оператора присваивания.

Пользовательские функции по функциональности похожи на хранимые процедуры. Разница заключается в том, что возможностей у них меньше (в частности, они должны возвращать только одно значение, например, скалярное или табличное), но их удобнее использовать с точки зрения синтаксиса.

Как процедуры, так и функции могут возвращать значения (в виде набора записей). Различие состоит в том, что функция вызывается из запроса, а процедура из отдельной команды.

На настоящий момент реализация хранимых процедур не поддерживает никаких внешних языков, но (по крайней мере, так заявляется) соответствует стандарту SQL:2003, позволяющему применять условные конструкции, итерации и обработку ошибок.

Пример создания хранимой процедуры в MySQL 5:

```
CREATE PROCEDURE p ()

LANGUAGE SQL

NOT DETERMINISTIC

SQL SECURITY DEFINER

COMMENT 'A Procedure' <--

SELECT CURRENT_DATE, RAND() FROM t
```

В данном случае мы создали процедуру с именем p, которая возвращает текущую дату и псевдослучайное число из таблицы t. Пример ее вызова и возвращаемого результата:

```
mysql> call p2()
+--------------+----------------+
| CURRENT_DATE | RAND()         |
+--------------+----------------+
| 2005-06-27   | 0.7822275075896 |
+--------------+----------------+
1 row in set (0.26 sec)
Query OK, 0 rows affected (0.26 sec)
```

Чуть более сложный пример создания и использования функции:

```
CREATE FUNCTION factorial (n DECIMAL(3,0))
RETURNS DECIMAL(20,0)
DETERMINISTIC
BEGIN
DECLARE factorial DECIMAL(20,0) DEFAULT 1;
DECLARE counter DECIMAL(3,0);
SET counter = n;
factorial_loop: REPEAT
SET factorial = factorial * counter;
SET counter = counter - 1;
UNTIL counter = 1
END REPEAT;
RETURN factorial;
END
```

В приложении:

```
INSERT INTO t VALUES (factorial(pi))
```

SELECT s1, factorial (s1) FROM t

UPDATE t SET s1 = factorial(s1)

WHERE factorial(s1) < 5

Разумеется эффективность применения хранимых процедур существенно возрастает при вызове их с параметрами (аргументами). Ниже дан пример процедуры с обработкой переданных ей параметров:

CREATE PROCEDURE p1 (IN parameter1 INT)

BEGIN

DECLARE variable1 INT;

SET variable1 = parameter1 + 1;

IF variable1 = 0 THEN

INSERT INTO t VALUES (17);

END IF;

IF parameter1 = 0 THEN

UPDATE t SET s1 = s1 + 1; <--

ELSE

UPDATE t SET s1 = s1 + 2;

END IF;

END;

Вызов процедуры теперь будет таким:

mysql> CALL p2(0) // Query OK, 2 rows affected (0.28 sec)

и в результате запроса мы получим:

```
mysql> SELECT * FROM t
+----+
| s1 |
+----+
| 6  |
```

```
| 6 |
+------+
2 rows in set (0.01 sec)
```

Кроме условных, возможны и любые циклические конструкции:

```
CREATE PROCEDURE p3 ()
BEGIN
DECLARE v INT;
SET v = 0;
WHILE v < 5 DO
INSERT INTO t VALUES (v);
SET v = v + 1;
END WHILE;
END;
```

Вызов процедуры:

```
mysql> CALL p3()
+------+
| s1   |
+------+
…….


|   0 |
|   1 |
|   2 |
|   3 |
|   4 |
+------+
Query OK, 1 row affected (0.00 sec)
```

Также применимы итерации, переходы, словом, всё, что предполагает стандарт.

Внутри функций и хранимых процедур осуществлена реализация курсоров, но, к сожалению, она пока ограничена (ASESITIVE, READ ONLY и NONSCROLL):

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
DECLARE a,b INT;
DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET b = 1;
OPEN cur_1;
REPEAT
FETCH cur_1 INTO a;
UNTIL b = 1
END REPEAT;
CLOSE cur_1;
SET return_val = a;
END;
```

Создание процедур и функций

CREATE

[DEFINER = { *user* | CURRENT_USER }]

PROCEDURE *sp_name* ([*proc_parameter*[,...]])

[*characteristic* ...] *routine_body*

CREATE

[DEFINER = { *user* | CURRENT_USER }]

FUNCTION *sp_name* ([*func_parameter*[,...]])

RETURNS *type*

[*characteristic* ...] *routine_body*

proc_parameter:

[IN | OUT | INOUT] *param_name type*

func_parameter:

param_name type

type:

Any valid MySQL data type

characteristic:

LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

| COMMENT '*string*'

routine_body:

Внесение изменений

ALTER {PROCEDURE | FUNCTION} *sp_name* [*characteristic* ...]

characteristic:
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT '*string*'

Удаление процедур и функций

DROP {PROCEDURE | FUNCTION} [IF EXISTS] *sp_name*

Вызов процедур и функций

CALL *sp_name*([*parameter*[,...]])
CALL *sp_name*[()]

Оператор CALL позволяет вызвать ранее определенную процедуру.

Пример1

```
CREATE PROCEDURE p1 (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
 # Set value of OUT parameter
 SELECT VERSION() INTO ver_param;
 # Increment value of INOUT parameter
 SET incr_param = incr_param + 1;
END;
```

Перед вызовом процедуры инициализируйте переменную указанные в параметрах INOUT . После вызова процедуры значения будут установлены или изменены.

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+----------------+------------+
| @version       | @increment |
+----------------+------------+
| 5.1.12-beta-log | 11         |
```

Пример2

```
CREATE PROCEDURE `p2`(IN param1 CHAR(2) )
  NOT DETERMINISTIC
```

```
    SQL SECURITY DEFINER
    COMMENT ''
  BEGIN
  select * from s where snum=param1;
  END;
```

Вызов процедуры
call p2 ('S1')

Пример3

```
  CREATE PROCEDURE `My_proc2`(IN param1 CHAR(2) )

  BEGIN                          /* start of block */
   DECLARE variable1 CHAR(10);          /* variables */
   IF param1 = 17 THEN              /* start of IF */
    SET variable1 = 'birds';            /* assignment */
   ELSE
    SET variable1 = 'beasts';           /* assignment */
   END IF;                        /* end of IF */
  select variable1;   /* statement */
  END
```

Вызов процедуры
call p3 (10)

Триггеры

Три́ггер (англ. trigger) — это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено наступлением определенного события (действием) — по сути добавлением INSERT или удалением DELETE строки в заданной таблице, или модификаци UPDATE данных в определенном столбце заданной таблицы реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции. Момент запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанного с ним события; например, до добавления записи) или AFTER (после события). В случае, если триггер вызывается до события, он может внести изменения в модифицируемую событием запись (конечно, при условии, что событие — не удаление записи). Некоторые СУБД накладывают ограничения на операторы, которые могут быть использованы в триггере (например, может быть запрещено вносить изменения в таблицу, на которой «висит» триггер, и т. п.)

Кроме того, триггеры могут быть привязаны не к таблице, а к представлению (VIEW). В этом случае с их помощью реализуется механизм «обновляемого представления». В этом случае ключевые слова BEFORE и AFTER влияют лишь на последовательность вызова триггеров, так как собственно событие (удаление, вставка или обновление) не происходит.

CREATE

[DEFINER = { user | CURRENT_USER }]

TRIGGER trigger_name trigger_time trigger_event

ON tbl_name FOR EACH ROW trigger_stmt

Пример создания и работы триггера:

CREATE TABLE t22 (s1 INTEGER)

CREATE TRIGGER t22_bi

BEFORE INSERT ON t22

FOR EACH ROW

BEGIN

SET @x = ‘Trigger was activated!’;

SET NEW.s1 = 55;

END;

После этого при выполнении запросов получим:

```
mysql> INSERT INTO t22 VALUES (1)
mysql> SELECT @x, t22.* FROM t22 // вызывается триггер
+----------------------+------+
| @x                   | s1   |
+----------------------+------+
| Trigger was activated! | 55   |
+----------------------+------+
1 row in set (0.00 sec)
```

Словарь данных

Иметь доступ к значениям метаданных – совершенно необходимое требование к современной СУБД. Ранее такая возможность в MySQL достигалась различными SHOW-командами, но такой подход имеет очевидные недостатки. Эти команды нельзя использовать в простых запросах с соединениями, и, что существенно, они не соответствовали стандартам, будучи специфичными для MySQL.

В новой версии СУБД появилась новая служебная база данных – INFORMATION_SCHEMA. Её наличие продиктовано тем же стандартом SQL:2003, и именно она решает задачу реализации словаря данных (data dictionary). INFORMATION_SCHEMA содержит таблицы, описывающие состояние и параметры сервера, в том числе определения и сущности таблиц. Это виртуальная база данных – физически (в виде файлов на диске) она не существует, вся информация динамически предоставляется сервером. Пример использования этой таблицы:

```
mysql> SELECT table_name, table_type, engine
    -> FROM INFORMATION_SCHEMA.tables
    -> WHERE table_schema = 'tp'
    -> ORDER BY table_type ASC, table_name DESC;
+-----------+-----------+--------+
| table_name | table_type | engine |
+-----------+-----------+--------+
| t2        | BASE TABLE | MyISAM |
| t1        | BASE TABLE | InnoDB |
```

```
| v1        | VIEW      | NULL   |
+-----------+-----------+--------+
```

Другой пример работы со словарём данных – просмотр привелегий:

```
mysql> SELECT * FROM
    -> INFORMATION_SCHEMA.COLUMN_PRIVILEGES\G
*************************** 1. row ***************************
GRANTEE: 'peter'@'%'
TABLE_CATALOG: NULL
TABLE_SCHEMA: tp
TABLE_NAME: t1
COLUMN_NAME: col1
PRIVILEGE_TYPE: UPDATE
IS_GRANTABLE: NO
*************************** 2. row ***************************
GRANTEE: 'trudy'@'%'
TABLE_CATALOG: NULL
TABLE_SCHEMA: tp
TABLE_NAME: t2
COLUMN_NAME: col1
PRIVILEGE_TYPE: SELECT
IS_GRANTABLE: YES
```

Объявление переменных

Объявление. DECLARE Local Variables

Следующая команда позволяет объявлять локальные переменные, содержит возможность задания значения по умолчанию. Переменная может быть объявлена как выражения, не обязательна константа. Если значение по умолчанию не определено то равно NULL.

DECLARE *var_name*[,...] *type* [DEFAULT *value*]

Присваивание Variable SET Statement

SET *var_name* = *expr* [, *var_name* = *expr*] ...

SELECT ... INTO Statement

Оператор SELECT может перенаправить результат в переменные. Таким образом может быть преобразована только одна строка.

ПРИМЕР

SELECT *col_name*[,...] INTO *var_name*[,...] *table_expr*
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;

Условия и ограничения

Объявление условий

DECLARE *condition_name* CONDITION FOR *condition_value*

condition_value:
  SQLSTATE [VALUE] *sqlstate_value*
 | *mysql_error_code*

Объявление ограничений

DECLARE *handler_type* HANDLER FOR *condition_value*[,...] *statement*

handler_type:
  CONTINUE
 | EXIT
 | UNDO

condition_value:
  SQLSTATE [VALUE] *sqlstate_value*
 | *condition_name*
 | SQLWARNING
 | NOT FOUND
 | SQLEXCEPTION

| *mysql_error_code*

Пример

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
    ->   SET @x = 1;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo()//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
    +------+
    | @x   |
    +------+
    | 3    |
    +------+
    1 row in set (0.00 sec)
```

Если вы хотите игнорировать условие вы должны сгенерировать ограничение и ассоциировать его с пустым блоком .

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

Пример

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  retry:
    REPEAT
      BEGIN
        DECLARE CONTINUE HANDLER FOR SQLWARNING
```

```
      BEGIN
        ITERATE retry;  # illegal
      END;
    END;
    IF i < 0 THEN
      LEAVE retry;       # legal
    END IF;
    SET i = i - 1;
  UNTIL FALSE END REPEAT;
END;
```

Курсоры

Курсор — в некоторых реализациях языка программирования SQL (Oracle, Microsoft SQL Server) — получаемый при выполнении запроса результирующий набор и связанный с ним указатель текущей записи.

Курсор может возвращать одну строку, несколько строк или ни одной строки. Для запросов, возвращающих более одной строки, можно использовать только явный курсор. Для повторного создания результирующего набора для других значений параметров курсор следует закрыть, а затем повторно открыть.

Курсор может быть объявлен в секциях объявлений любого блока PL/SQL, подпрограммы или пакета.

Операторы управления явным курсором

- Оператор CURSOR выполняет объявление явного курсора.
- Оператор OPEN открывает курсор, создавая новый результирующий набор на базе указанного запроса.
- Оператор FETCH выполняет последовательное извлечение строк из результирующего набора от начала до конца.
- Оператор CLOSE закрывает курсор и освобождает занимаемые им ресурсы

Курсоры поддерживают хранимые процедуры и функции. Сейчас курсоры имеют три свойства:

- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable
- Non-scrollable: Can be traversed only in one direction and cannot skip rows

Курсоры должны быть объявлены перед объявлением ограничений. Переменные и условия должны быть объявлены перед курсором.

Объявление курсоров

Оператор объявления курсора. М программе можно объявлять несколько курсоров, каждый курсор в блоке должен иметь уникальное имя.

DECLARE *cursor_name* CURSOR FOR *select_statement*

Условие открытия Cursor OPEN Statement

Оператор открывает ранее объявленный курсор.

OPEN *cursor_name*

Выполнение курсора Cursor FETCH Statement

FETCH *cursor_name* INTO *var_name* [, *var_name*] ...

Условия закрытия Cursor CLOSE Statement

CLOSE *cursor_name*

Пример

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE a CHAR(16);
  DECLARE b,c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur1;
  OPEN cur2;

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
      IF b < c THEN
        INSERT INTO test.t3 VALUES (a,b);
      ELSE
        INSERT INTO test.t3 VALUES (a,c);
      END IF;
    END IF;
  UNTIL done END REPEAT;
```

```
  CLOSE cur1;
  CLOSE cur2;
END
```

Лабораторная работа 3.1

Задание

Представления

1. Составить представление (не менее 5 представлений для своего индивидуального задания).

Курсоры

2. Создать несколько курсоров (не менее 2 некластерных и один кластерный для своего индивидуального задания).

Процедуры

3. Создать 4 функции пользователя, на выбор для своего индивидуального задания.

Лабораторная работа 3.2

1. Составить процедуру (не менее 5 процедур для своего индивидуального задания).

Создание триггеров

Создать 4 триггера для своего индивидуального занятия:

1 Триггер INSERT;

2 Триггер DELETE;

3 Триггер UPDATE;

4 Триггер INSTEAD OF.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Информация и данные, база данных, система управления базами данных (СУБД).
2. Эволюция концепции обработки данных, СУБД.
3. Требования к СУБД, основные особенности СУБД, составные части СУБД,
4. Уровни абстрагирования процессов обработки данных: инфологическая, даталогическая и физическая модели данных.
5. Виды даталогических моделей, принципы организации данных, достоинства и недостатки.
6. Реляционная БД, история появления, принципы организации данных, достоинства и недостатки.
7. Базовые понятия реляционных БД: тип данных, домен, атрибут, кортеж, отношение, схема отношений.
8. Понятие ключа: первичный и внешний ключи, простой ключ, составной ключ, ключи-кандидаты, требования к первичному ключу
9. Определение реляционной модели данных согласно Дейту, структурная, манипуляционная и целостная части модели, 2 базовых требования целостности и способы их реализации.
10. Проектирование БД с использованием нормализации, 5 видов нормальных форм, общие свойства нормальных форм.
11. Нормализация БД, цели нормализации, 1НФ, понятие атомарности значений.
12. Нормализация БД, цели нормализации, 2НФ, понятие функциональной зависимости, полной функциональной зависимости, многозначной зависимости.
13. Третья нормальная форма, понятие зависимости и транзитивной зависимости, переход от 2НФ к 3 НФ.
14. Нормализация БД, определение 1НФ, 2НФ, 3НФ, 4НФ, 5НФ.
15. Обънетно-ориентированное программрование в СУБД.
16. Типичная структура файла БД на примере DBF, понятие логического и физического удаления записей, смысл операции упаковки базы..
17. Сетевые БД, архитектура «файл-сервер», «клиент-сервер», «толстый» и «тонкий» клиент.
18. Сетевые БД, распределенная архитектура сетевых БД, сущность репликации БД.
19. Понятие транзакции, принципы работы транзакций, роль транзакций в многопользовательской и однопользовательской СУБД.
20. Понятие транзакции, принципы работы транзакций, три уровня изолированности транзакций, сериализация транзакций.
21. Журнализация действий пользователя, журнализация транзакций, виды журналов.
22. Язык SQL: общие сведения о языке, роль и место в современных СУБД, стандарт ANSI, «диалекты» (на примере Jet SQL).
23. Запрос выборки данных в SQL, простейшая выборка из одной таблицы, секция WHERE, специальные операторы условия.
24. SQL: запрос выборки данных, функции агрегирования AVG, SUM, MAX, MIN.
25. SQL: запрос выборки данных, секции GROUP BY и HAVING.
26. SQL: запрос выборки данных, использование констант и выражений, упорядочение данных.
27. SQL: запрос выборки данных по нескольким таблицам, соединение таблиц с помощью секции WHERE, задание псевдонимов (alias) и их использование.
28. SQL: запрос выборки данных по нескольким таблицам, оператор JOIN, левое, правое и внутреннее соединение.
29. SQL: вложенные подзапросы, обеспечение единственности или множественности возвращаемых подзапросом значений (DISTINCT).

30. SQL: связанные подзапросы, определение, порядок выполнения, область применения.
31. SQL: связанные подзапросы, служебные операторы EXISTS, ANY, SOME, ALL.
32. SQL: объединение запросов оператором UNION.
33. SQL: запросы обновления таблиц INSERT, UPDATE, DELETE.
34. SQL: создание новой таблицы на основе существующей таблицы, изменение структуры существующей таблицы, удаление таблицы.
35. SQL: создание новой таблицы командами языка описания данных, типы данных, типы ограничений.
36. SQL: задание связей между таблицами, задание ключей, создание и удаление индексов.
37. SQL: хранимые процедуры, описание, область применения.
38. SQL: триггеры, особенности применения, момент и порядок выполнения.
39. SQL: представления (view), объявление, область применения, признаки изменяемых представлений
40. SQL: курсоры, объявление, область применения
41. SQL: права доступа к данным, виды привилегий, системные и объектные привилегии
42. Создание базы данных.
43. Создание базы данных в среде MS SQL Server.
44. Изменение базы данных.
45. Удаление базы данных.
46. Создание таблицы.
47. Изменение таблицы.
48. Удаление таблицы.
49. Индексы в стандарте языка SQL.
50. Индексы в среде MS SQL Server.
51. Создание индекса.
52. Некластерный и кластерный индексы.
53. Уникальный индекс.
54. Удаление индекса.
55. Понятие функции пользователя.
56. Функции Scalar.
57. Функции Inline.
58. Функции Multi-statement.
59. Встроенные функции.
60. Функции для работы с датой и временем.
61. Понятие хранимой процедуры.
62. Хранимые процедуры в среде MS SQL Server.
63. Создание, изменение и удаление хранимых процедур.
64. Выполнение хранимой процедуры.
65. Понятие курсора.
66. Реализация курсоров в среде MS SQL Server.
67. Управление курсором в среде MS SQL Server.
68. Изменение и удаление данных с помощью курсора.
69. Закрытие и освобождение курсора.
70. Определение триггера в стандарте языка SQL.
71. Реализация триггеров в среде MS SQL Server.
72. Типы триггеров.
73. Программирование триггера.
74. Введение в транзакции.

75. ACID-свойства транзакций.
76. Блокировки.
77. Управление транзакциями.
78. Управление транзакциями в среде MS SQL Server.
79. Блокировки в среде MS SQL Server.
80. Управление пользователями базы данных.
81. Управление пользователями в среде MS SQL Server.
82. Управление доступом к данным.
83. Реализация прав на доступ к объектам баз данных в среде MS SQL Server.
84. Сравнительный анализ серверов баз данных.
85. Основные отличия MS SQL – сервера от MY SQL

ТЕСТЫ

Основные понятия и технологии работы с реляционными базами данных/Понятие систем управления базами данных

1. **СУБД работающие с какой моделью данных получили самое широкое распространение:**

- ○ Списки
- ● Реляционной
- ○ Сетевой
- ○ Иерархической

2. **Основное назначение менеджера транзакций:**

- ○ Управление оперативной памятью
- ○ Найти лучший способ выполнения требуемой операции и дать соответствующие команды менеджеру памяти
- ○ Контролирует расположение файлов на диске
- ● Гарантировать правильное выполнение всех транзакций

3. **Основные требования, предъявляемые к выполнению транзакций (отметьте четыре варианта):**

- ● **25%** Атомарность
- ● **25%** Непротиворечивость
- ● **25%** Изоляция
- ● **25%** Долговременность
- ○ **-100%** Запрет на взаимодействие с запросами

4. **Подсистема обработки применяется для:**

- ○ Выполнения функции посредника между подсистемой средств проектирования и обработки и данными
- ● Обработки компонентов приложений, созданных с помощью средств проектирования
- ○ Упрощения проектирования и реализации баз данных и их приложений

5. **Для чего служат метаданные?**

- ○ Управление основной памятью
- ● Хранение информации о структуре данных
- ○ Обработки сбоев
- ○ Упрощение проектирования

6. **По степени универсальности различают СУБД (отметьте два варианта):**

- -100% Сетевые
- 50% Общего назначения
- -100% Реляционные
- -100% Иерархические
- 50% Специального назначения

7. Какой язык используется для работы в современных СУБД?

- DML
- SDL
- SQL

8. Какие СУБД относятся к категории персональных? (отметьте два варианта)

- 50% FoxBase
- -100% Sybase
- 50% FoxPro
- -100% Inqres

9. СУБД это:

- Часть реального мира, подлежащая изучению с целью создания базы данных для автоматизации процесса управления
- Любой конкретный или абстрактный объект в рассматриваемой предметной области.
- Это свойство сущности в предметной области
- Это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями

10. Механизм запросов используется для:

- Добавление записей
- Удаление записей
- Написания сторонних приложений
- Обновления записей

Основные понятия и технологии работы с реляционными базами данных/Модели ранних СУБД

11. Достоинства ранних СУБД (отметьте три варианта):

- 33% Развитые средства управления данными во внешней памяти на низком уровне
- 33% Возможность построения вручную эффективных прикладных систем
- 34% Возможность экономии памяти за счет разделения подобъектов (в сетевых системах)

- -100%Прикладные системы зависят от этой организации

12. Недостатки ранних СУБД (отметьте три варианта):

- 33%Слишком сложно пользоваться
- 33%Их логика перегружена деталями организации доступа к БД
- 34%Фактически необходимы знания о физической организации
- -100%Возможность построения вручную эффективных прикладных систем

13. Какая модель данных изображена на рисунке:

- Сетевая модель
- Иерархическая модель
- Реляционная модель данных
- Информационно-логическая модель данных

14. Разница между иерархической моделью данных и сетевой состоит:

- В том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков
- В том, что в сетевых структурах запись-потомок должна иметь в точности одного предка, а в иерархической структуре данных у потомка может иметься любое число предков

15. Какая модель данных является самой распространенной в настоящее время?

- Сетевая модель
- Иерархическая модель
- Реляционная модель данных
- Информационно-логическая модель данных

16. Почему СУБД компании Microsoft получили наибольшее распространение?

- Они имеют большие возможности интеграции, совместной работы и использования данных, так как данные пакеты являются продуктами одного производителя, а также используют сходные технологии обмена данными
- Из-за небольшой стоимости
- Из-за большой популярности компании Microsoft

17. Укажите СУБД компании Microsoft (отметьте два варианта):

- -100%Oracle
- 50%SQL Server
- 50%Visual FoxPro
- -100%Lotus Approach
- -100%dBase

18. Технология «Клиент-сервер» изображена на рисунке?

- 

- 

-

19. Администрирование базы данных это:

- Это функция управления базой данных
- Это лицо, отвечающее за выработку требований к базе данных, её проектирование, реализацию, эффективное использование и сопровождение

Основные понятия и технологии работы с реляционными базами данных/Реляционная модель данных

20. Какой из модели данных соответствует данное утверждение: “Данные в БД представляют собой набор отношений”?

- ○ Сетевая модель данных
- ● Реляционная модель данных
- ○ Иерархическая модель данных
- ○ Не соответствует ни одной из моделей данных

21. Что характерно для реляционной модели данных? (отметьте три варианта)

- ● **33%** Использования понятия отношений и таблиц
- ○ **-100%** Древовидная структура
- ● **33%** Модель является логической
- ○ **-100%** Модель является физической
- ○ **-100%** Характеризуется такими понятиями как уровни, узлы и связи
- ● **34%** Использование теории нормализации

22. Какие из перечисленных объектов можно отнести к объектам реляционной модели данных? (отметьте три варианта)

- ● **33%** Кортеж
- ○ **-100%** Узел
- ○ **-100%** Уровень
- ● **33%** Таблица
- ● **34%** Отношение
- ○ **-100%** Предок

23. Что такое кортеж? Выберите верное утверждение.

- ● Это элемент отношения, строка таблицы; упорядоченный набор из N элементов
- ○ Поле, элемент данных
- ○ Отношение между таблицами
- ○ Ни одно из утверждений не верно

24. Какие утверждения верны для таблицы в реляционной БД? (отметьте три варианта)

- ○ **-100%** Могут присутствовать одинаковые строки
- ● **33%** Все ячейки в столбце однородны
- ○ **-100%** Ячейки в столбце могут быть неоднородными
- ● **33%** Порядок следования строк в таблице произволен
- ● **34%** Каждый элемент таблицы (строка) – один элемент данных

- -100%Порядок следования строк строго определен
- -100%Ни одно из утверждений не верно

25. Почему следует выполнять условие целостности данных?

- Для удобства
- Обязательное требование, необходимое для корректного функционирования БД
- Условие целостности не обязательно соблюдать

26. Что из себя представляет целостность по существованию?

- Целостность по существованию определяется понятием внешнего ключа, и определяет набор правил для поддержания целостности и связей между таблицами
- Между таблицами должны существовать связи
- Потенциальный ключ отношения не может принимать значение NULL
- Нет верного утверждения

27. Что такое первичный ключ?

- Потенциальный ключ, который берется в качестве основного ключа, и который однозначно и уникально характеризует запись в таблице
- Первое поле в таблице
- Поле, которое ссылается на ключ внешней таблицы
- Верного утверждения нет

28. Почему рекомендуется использовать суррогатные ключи, а не естественные? (отметьте два варианта)

- 50%У естественных ключей низкая эффективность
- -100%Суррогатные ключи повторяются
- 50%Легче проводить каскадное обновление таблиц
- -100%Суррогатные ключи не рекомендуется использовать

29. Благодаря чему устанавливается связь между отношениями?

- Ссылкам
- Альтернативным ключам
- Первичному ключу
- Внешнему ключу

30. Использования внешних ключей позволяет (отметьте два варианта):

- 50%Организовать связи между отношениями
- 50%Избежать дублирования данных
- -100%Упростить структуру БД

- -100%Ни одно из утверждений не верно

31. Пользователю необходимо удалить запись из родительской таблицы, причем у нее имеются подчиненная таблица, в которой используется эта запись. Чтобы удалить запись в родительской таблице, при этом не удаляя записи с этим внешним ключом в подчиненной, какие правила можно установить? (отметьте два варианта)

- -100%Restrict
- -100%Cascade
- 50%Set Null
- 50%Set Default

32. Чтобы выполнялось обновление, как в родительской таблице, так и в подчиненных, необходимо установить правило:

- Restrict
- Cascade
- Set Null
- Set Default

Основные понятия и технологии работы с реляционными базами данных/Наиболее распространенные виды моделей данных

33. Иерархическая модель данных — логическая модель данных в виде:

- Двоичное дерево поиска
- Очередь с приоритетом
- Древовидной структуры
- Ассоциативного массива

34. Назовите достоинства иерархической модели данных (отметьте два варианта):

- 50%Эффективное использование памяти ЭВМ
- -100%Громоздкость для обработки информации с достаточно сложными логическими связями
- -100%Сложность понимания для обычного пользователя
- 50Неплохие показатели времени выполнения основных операций над данными
- -100%Нет правильных ответов

35. Назовите недостатки иерархической модели данных (отметьте два варианта):

- -100%Нет возможности работать по сети
- 50%Сложность понимания для обычного пользователя
- -100%Эффективное использование памяти ЭВМ

- **50%**Громоздкость для обработки информации с достаточно сложными логическими связями
- **-100%**Нет правильных вариантов ответов

36. Сетевая модель данных — логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных. Разница между иерархической моделью данных и сетевой состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься:

- Любое число предков
- Один предок
- Один потомок
- Нет правильного ответа

37. Назовите достоинства сетевой модели данных:

- Сложность понимания для обычного пользователя
- Эффективное использование памяти ЭВМ
- Высокая сложность и жесткость схемы БД, построенной на ее основе
- Возможность эффективной реализации по показателям затрат памяти и оперативности

38. Назовите недостатки сетевой модели данных:

- Нет возможности работать по сети
- Неплохие показатели времени выполнения основных операций над данными
- Высокая сложность и жесткость схемы БД, построенной на ее основе
- Эффективное использование памяти ЭВМ

39. В реляционной модели данных объекты и взаимосвязи между ними представлены в виде:

- Столбцов
- Графов
- Таблиц
- Деревьев

40. Назовите недостатки реляционной модели данных (отметьте три варианта):

- **33%**По сравнению с иерархической и сетевой моделями реляционная модель имеет более низкую скорость доступа и требует большего объема внешней памяти. В настоящее время этот фактор не является критическим вследствие многократно возросшего быстродействия компьютеров и такого же роста объема дисковой памяти
- **33%**Часто в результате логического проектирования появляется очень много таблиц, что затрудняет понимание структуры данных

- **34%** Далеко не всегда предметную область можно представить в виде совокупности таблиц. Так, в системах автоматизации проектирования и автоматизированной разработки программного обеспечения требуются гораздо более сложные структуры данных
- **-100%** Сложность понимания для обычного пользователя
- **-100%** При проектировании реляционных баз данных применяются строгие правила, базирующиеся на математическом аппарате
- **-100%** Нет правильных вариантов

41. Назовите достоинства реляционной модели данных (отметьте четыре варианта):

- **-100%** Нет правильных вариантов ответа
- **-100%** Нет возможности работать по сети
- **-100%** Сложность понимания для обычного пользователя
- **25%** Одним из важных достоинств реляционного подхода является его простота и доступность для понимания конечным пользователем. Единственной информационной конструкцией является таблица
- **25%** При проектировании реляционных баз данных применяются строгие правила, базирующиеся на математическом аппарате
- **25%** Реляционная модель обеспечивает полную независимость данных. При изменении структуры реляционной базы данных изменения, которые требуется произвести в прикладных программах, как правило, минимальны
- **25%** Манипулирование данными на уровне языка СУБД производится ненавигационно, поэтому для построения запросов и написания прикладных программ нет необходимости знания конкретной организации базы данных во внешней памяти. Конечно, при исполнении запросов на физическом уровне выполняется навигация по записям таблиц, однако эти действия производятся процедурами самой СУБД

42. Для преодоления недостатков, присущих реляционной модели, в настоящее время развивается:

- Сетевая модель
- Иерархическая модель
- Постреляционная модель
- Нет правильных вариантов

Основные понятия и технологии работы с реляционными базами данных/Математический аппарат реляционных БД

43. Операции реляционной алгебры делятся на (отметьте два варианта):

- **50%** Основные
- **-100%** Логические
- **50%** Дополнительные

- -100%Арифметические

44. В реляционной алгебре операнды и результаты всех операций являются:

- Отношениями
- Выражениями
- Аргументами

45. В контексте баз данных реляционное исчисление существует в двух формах:

- 50%В форме реляционного исчисления кортежей
- 50%В форме реляционного исчисления доменов
- -100%В форме реляционного исчисления выражений
- -100%В форме реляционного исчисления атрибутов

46. Реляционное исчисление кортежей состоит в отыскании таких кортежей, для которых:

- Предикат является ложным
- Предикат неопределен
- Предикат является истинным

47. Все запросы, которые можно сформулировать с помощью реляционной алгебры:

- Нельзя сформулировать с помощью реляционного исчисления и наоборот
- Можно сформулировать с помощью реляционного исчисления и наоборот
- Не всегда можно сформулировать с помощью реляционного исчисления и наоборот

48. Реляционная алгебра в отличии от реляционного исчисления:

- Задает порядок операций
- Оставляет компилятору определять наиболее эффективный порядок вычисления

49. К реляционным операторам относятся (отметьте два варианта):

- -100%And
- 50%= (Равно)
- -100%Or
- 50%< > (Не равно)

50. Три основные команды DML:

- INSERT, UPDATE, DELETE
- UNION, EXISTS, CREAT
- INSERT, CREAT, EXISTS

51. В каком из вариантов правильно записан формат команды Insert:

- Insert Into имя_таблицы [(список_имен)]
  Values (список_значений);
- Insert Into имя_таблицы [(список_имен)]
  Set поле = значение, [поле = значение], …
  Where условие;
- Insert Into имя_таблицы [(список_имен)]
  Where условие;

52. Команды INSERT, UPDATE, DELETE называются в SQL:

- Командами обновления данных
- Командами выборки данных
- Командами объединения таблиц
- Командами управления БД

Работа с современными промышленными СУБД/Языки управления базами данных

53. К языку манипулирования данными не относится команда:

- Вставить
- Создать
- Обновить
- Удалить

54. Семейство компьютерных языков, используемых в компьютерных программах или пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных:

- ЯМД
- ЯОД
- SQL
- СУБД

55. Какая команда предназначена для изменения данных в таблицах?

- Change
- Direct
- Alter
- Turn

56. В каком году был принят первый стандарт SQL?

- 1992
- 1978
- 1999

- 1986

57. Какая команда предназначена для удаления таблицы?

- DELETE TABLE
- DROP TABLE
- KILL TABLE
- CRASH TABLE

58. Можно ли создать таблицу, используя бланк QBE?

- Да
- Нет
- Зависит от версии MS Access

59. ::Тип SQL:: Какой тип SQL применяется для выполнения действий непосредственно в БД, чтобы получить результат?

- Интерактивный

60. Язык SQL позволяет выполнить следующие виды запросов (отметьте три варианта):

- 33%Обновления данных
- 33%Создания таблицы
- -100%Отправка данных
- 34%Управление БД

61. Какие типы языка SQL из перечисленных существуют (отметьте два варианта)?

- -100%Внешний
- 50%Встроенный
- 50%Интерактивный
- -100%Интегрированный

62. Какая команда служит для уничтожения представлений?

- DROP VIEW
- DELETE VIEW
- DROP LOOK
- DELETE KEY

Работа с современными промышленными СУБД/Организация данных в СУБД

63. При физической организации данных записи в файлах могут быть (отметьте два варианта):

- 50% Фиксированной длины
- -100% Произвольной адресации
- 50% Произвольной длины
- -100% Последовательной адресации

64. Для файлов последовательного доступа применяются следующие механизмы позиционирования (отметьте два варианта):

- -100% Указание индексов
- 50% Указание в начале записи ее длины
- 50% Указание специальным образом конца записи
- -100% Все варианты неверны

65. Недостатки метода хэширования (отметьте два варианта):

- 50% Возможность ситуации, когда для разных ключей будет получено одно и то же значение хэш-функции
- -100% Возможность ситуации, когда для разных ключей будут получены разные значения хэш-функции
- 50% Не всегда удается найти подходящую хэш-функцию
- -100% Значение хэш-функции зависит от первичного ключа

66. Несмотря на высокую эффективность метода хэширования, не всегда удается найти подходящую хэш-функцию. Поэтому для организации доступа по первичному ключу часто используют:

- Индексные файлы
- Файлы прямого доступа
- Функцию, однозначно связывающую номер записи и значение первичного ключа
- Файлы последовательного доступа

67. Достоинства хранения информации СУБД на «чистых» дисках (отметьте два варианта):

- -100% Использование стандартных средств обслуживания файлов
- 50% Внешняя память используется более эффективно
- 50% Как правило, увеличивается производительность обмена с дисками
- -100% Все варианты неверны

68. Достоинства работы СУБД с дисками через файловую систему (отметьте два варианта):

- -100% Экономия внешней памяти
- 50% В некоторых случаях выполнение операций ввода/вывода через файловую систему обеспечивает оптимизацию, которую СУБД не может реализовать

- 50%Использование файловой системы обладает большей гибкостью
- -100%Все варианты верны

69. Независимость данных – это:

- Неискажение данных при работе в многопользовательском режиме и в распределенных базах данных
- Адекватность отображения данных соответствующей предметной области
- Возможность изменения логической и физической структуры БД без изменения представлений пользователей
- Устойчивость хранимых данных к разрушению и уничтожению

70. Независимость данных (отметьте два варианта):

- 50%Предполагает инвариантность к характеру хранения данных
- -100%Предполагает защиту от ошибок при обновлении БД
- -100%Обеспечивает совместное использование данных многими пользователями
- 50%Обеспечивает минимальные изменения структуры БД при изменениях стратегии доступа к данным

71. Нарушение целостности данных может быть вызвано:

- Совместным выполнением конфликтных запросов пользователей
- Ошибками санкционированных пользователей или умышленные действия несанкционированных пользователей
- Программными сбоями СУБД или ОС
- Все варианты верны

72. Защита данных от несанкционированного доступа может достигаться (отметьте два варианта):

- 50%Получением разрешений от администратора базы данных
- -100%Защитой от ошибок при обновлении БД
- 50%Формированием видов - таблиц, производных от исходных и предназначенных конкретным пользователям
- -100%Стандартизацией построения и эксплуатации БД

Работа с современными промышленными СУБД/Методы поиска и анализа информации в базе данных

73. Какое выражение позволяет выполнить сортировку в обратном порядке?

- Order by desc;
- Sort by desc;
- Order by asc;

- Sort by asc;

74. Чем характеризуется каждая запись в таблице?

- Типом данных
- Своим порядковым номером
- Номером строки

75. Куда добавляется новая запись в таблице?

- В начало
- В конец
- По выбору
- Не добавляется

76. Упорядоченный список содержимого столбцов или группы столбцов - это?

- Дерево
- Индекс
- Модуль
- Адрес

77. Недостатком индексирования является?

- Индексирование проходит очень медленно
- Таблицу индексов необходимо перестраивать всякий раз, когда происходит изменение данных в таблице
- Индексирование проходит по дереву индексов
- При создании составного индексного файла индексирование выполняется не по одному, а по нескольким полям

78. Укажите, признаки соответствующие работе индексирования (отметьте три варианта)

- **33%** Индексирование применяется для ускорения поиска
- **-100%** Индексирование замедляет работу поиска
- **33%** Простой индексный файл содержит в себе выровненное двоичное дерево
- **34%** Применение индексирования позволяет вести ускоренный поиск по разным полям таблицы, если для них построены индексы
- **-100%** Таблицу индексов не нужно перестраивать всякий раз, когда происходит изменение данных в таблице

79. Как организовано хранение индексов в MS Access и в FoxPro?

- В FoxPro для каждой таблицы индексов создаётся отдельный файл, а в MS Access индексы хранятся в том же файле, что и таблицы данных

- В MS Access для каждой таблицы индексов создаётся отдельный файл, а в FoxPro индексы хранятся в том же файле, что и таблицы данных
- В MS Access и FoxPro индексы хранятся в том же файле, что и таблицы данных
- В MS Access и FoxPro для каждой таблицы индексов создаётся отдельный файл

80. Укажите существующие виды индексов?

- Простые и сложные
- Кластерные и некластерные
- Ограниченные и неограниченные
- Последовательные и непоследовательные

81. На что необходимо обратить внимание при создании индекса?

- На дополнительную затрату ресурсов компьютера на сопровождение индекса
- На производительность системы в целом
- На удобность и надежность эксплуатации
- На модель данных

82. Что такое редкий индекс?

- Это файл с указателем
- Это файл с последовательностью пар ключей
- Это файл с последовательностью пар ключей и указателей
- Это файл с ключом

83. По какой команде назначается привилегии на доступ к собственной таблице?

- WITH GRANT
- GRANT ON OPTION
- SELECT FROM
- GRANT SELECT TO

84. Какая из привилегий даёт право модифицировать структуру таблицы?

- ALTER
- INSERT
- GRANT
- REFERENTS

85. Что позволяет выполнять привилегия EXECUTE?

- Предоставляет право создавать таблицы и индексы
- Пользователь с этой привилегией может удалить строки из таблицы
- Позволяет выполнять хранимую процедуру или функцию

- Пользователь с этой привилегией может выполнять запросы для таблицы
- Разрешает пользователю выполнять действия администратора базы данных, т.е. распоряжаться ею как своей собственной.

86. Какая команда осуществляет выборку информации?

- SELECT <список полей>
- <имя поля> IN <назначение>
- <имя поля> LIKE «шаблон»
- FROM <список таблиц>

87. Какой параметр определяет условное выражение, которое должно удовлетворять строки включаемые в таблицу результат?

- BETWEEN
- LIKE
- DISTINCT
- WHERE

Работа с современными промышленными СУБД/СУБД MS Access. Работа в интерактивном режиме

88. Выберите пункт, в котором соотношение тип поля – размер задано неверно:

- Текстовый – 255 байт
- Дата/время – 8 байт
- Денежный – 4 байта
- Memo – 64 000 байт

89. Какой тип поля обеспечивает связь и внедрение объектов, созданных в других приложениях?

- Счётчик
- Memo
- Объект
- Текстовый

90. Что такое маска поля?

- Описательное имя, которое будет использоваться вместо имени поля в отчётах
- Значение, которое присваивается полю при создании новой записи
- Шаблон, определяющий формат ввода значения поля
- Логическое выражение которое проверяется при вводе и редактировании поля

91. От чего зависят дополнительные параметры и ограничения, устанавливаемые для полей?

- От типа поля
- От связи между таблицами
- От заданных пользователем настроек
- От версии СУБД

92. Какой из типов связей является основным?

- 1:1
- 1:∞
- ∞:∞
- Все вышеперечисленные типы являются актуальными

93. Какой вид связи не допускается в реляционных БД?

- 1:1
- 1: ∞
- ∞:∞
- Все перечисленные виды допускаются

94. Что в MS Access является объектом для хранения данных?

- Запись
- Строка
- Таблица
- Столбец

95. Что является обязательным условием при создании связей между полями?

- Типы связываемых полей должны быть одинаковыми
- Связываемые поля должны иметь тип счётчик
- Связываемые поля должны иметь одинаковые имена
- Связываемые поля должны находиться в одной таблице

Работа с современными промышленными СУБД/СУБД MS Access. Связывание и фильтрация данных

96. С помощью какого инструмента наиболее удобно заполнять базу данных в MS Access?

- Отчёт
- Форма
- Конструктор
- Мастер

97. Каким образом работает Фильтр по выделенному?

- ○ Сохраняет в файл выделенные записи
- ○ На экран выводится пустая таблица или форма для активного объекта базы данных
- ● Извлекает из таблицы и выдает на экран только те записи, которые содержат выделенное значение
- ○ В верхней части выводится список полей активной таблицы. В нижней части окна выводится бланк запроса. В строку бланка запроса Поле перетаскиваются мышью из списка поля, по которым надо задать условия отбора записей. Условия отбора вводятся в соответствующую строку. Кроме того, в бланке запроса в строке Сортировка может быть выбран тип сортировки для одного или нескольких выбранных полей

98. Каким образом работает Обычный фильтр?

- ● После выполнения команды Записи|Фильтр|Изменить фильтр в окне обычного фильтра Фильтр на экран выводится пустая таблица или форма для активного объекта базы данных. На вкладке Найти в поля фильтра вводятся значения, по которым будут отбираться записи. Ввод значений в несколько полей одной строки фильтра определяет отбор записей, в которых присутствуют все указанные значения. При этом заданные условия рассматриваются как объединяемые логической операцией "И"
- ○ Извлекает из таблицы и выдает на экран только те записи, которые содержат выделенное значение
- ○ Обычно работает
- ○ Такого фильтра нет

99. Какие три вида фильтров предусмотрены в MS Access?

- ● **33%** Фильтр по выделенному
- ● **33%** Обычный фильтр
- ○ **-100%** Масляный фильтр
- ● **34%** Расширенный фильтр
- ○ **-100%** Фильтр с параметрами
- ○ **-100%** Фильтр Гейтса

100. Каким образом можно получить доступ к инструменту Схема данных?

- ○ Вид|Схема данных
- ● В окне Базы данных с выполнения команды Сервис|Схема данных
- ○ Файл|Схема данных
- ○ Такого инструмента нет в среде MS access

101. Сколько листов данных, за одну операцию импорта, можно импортировать из Exel в Access?

- ● Один
- ○ Два
- ○ Три

- Четыре

102. Какие виды связей между таблицами, можно создавать в MS Access? (отметьте два варианта)

- **50%** 1 к 1
- **50%** 1 к ∞
- **-100%** 1 к 2
- **-100%** 1 к 50

103. Как можно распечатать данные из базы данных?

- С использованием отчёта
- Через конструктор
- В MS Access нельзя вывести на печать документ
- Необходимо экспортировать данные в MS Word и только потом распечатать

104. Продолжите предложение. Если поле связи является уникальным ключом в одной таблице (главной таблицы связи), а в другой таблице (подчиненной таблице связи) является не ключевым или входит в составной ключ, то его значения …..

- Не могут повторятся
- Равны нулю
- Могут повторяться
- Суммируются

105. Технология связывания и внедрения объектов в другие документы и объекты, разработанные компанией Microsoft?

- PDM
- OLE
- OSPF
- STP

Работа с современными промышленными СУБД/Средства диалогового постороения запросов

106. Назовите основные преимущества перекрестных запросов:

- Возможность сортировки таблицы результатов по значениям, содержащимся в столбцах; простота и скорость разработки сложных запросов с несколькими уровнями детализации
- Простота и скорость разработки сложных запросов с несколькими уровнями детализации; возможность обработки значительного объема данных и вывода их в формате, который очень хорошо подходит для автоматического создания графиков и диаграмм

- Возможность обработки значительного объема данных и вывода их в формате, который очень хорошо подходит для автоматического создания графиков и диаграмм; простота построения отчётов
- Возможность сортировки таблицы результатов по значениям, содержащимся в столбцах; простота построения отчётов

107. Что такое схема данных?

- Схема: наглядно отображающая таблицы и связи между ними; обеспечивающая использование связей при обработке данных
- Схема, наглядно отображающая запросы и отчёты
- Схема: обеспечивающая использование связей при обработке данных; наглядно отображающая запросы
- Схема: наглядно отображающая таблицы и связи между ними; отображающая отчёты

108. Что позволяют выполнять запросы действий?

- Создавать таблицы, модифицировать данные в таблицах: удалять, обновлять, добавлять записи
- Удалять и добавлять данные в таблицах
- Создавать таблицы без возможности модифицировать данные в них
- Обновлять и добавлять записи в таблицах

109. Какие запросы относятся к запросам группы действий?

- Запросы на: создание таблиц, добавление, обновление, удаление записей в таблицах
- Запросы на создание отчётов
- Запросы на создание таблиц и отчётов
- Запросы на создание отчётов и удаление записей в таблицах

110. Что нужно сделать для того, чтобы создать многотабличный запрос?

- В окно конструктора запросов добавить все участвующие в выборке таблицы, определить условия объединения таблиц
- В окно конструктора запросов можно не добавлять все участвующие в выборке таблицы, но обязательно определить условия объединения таблиц
- В окно конструктора запросов добавить все участвующие в выборке таблицы
- В окно конструктора запросов добавить любой существующий запрос на выборку

111. Какая конструкция используется для группировки данных в запросе?

- Group by
- Order by
- Where
- Count
- Select

112. Какая строка отсутствует в QBE бланке:

- Поле
- Условия отбора
- Сортировка
- Обновление данных

113. Выберите верное высказывание:

- QBE - запрос по образцу – средство для отыскания необходимой информации в базе данных
- QBE – запрос формируется на специальном языке
- Все запросы Access строит на основе QBE – запросов
- В MS Access применяется только один тип запросов: по образцу (QBE – Query by example)

114. Какие поля необходимо определить для перекрестного запроса?

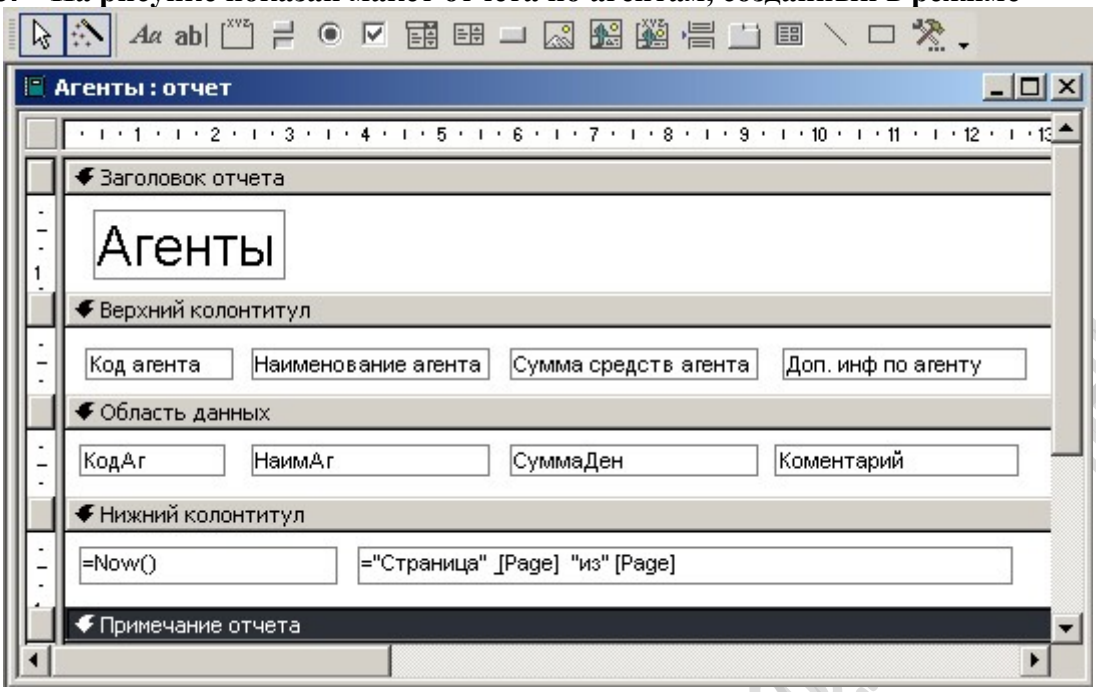
- Одно поле в качестве заголовков строк, одно для заголовков столбцов и одно поле значений
- Два поля в качестве заголовков строк, два для заголовков столбцов и одно поле значений
- Одно поле в качестве заголовков строк, одно для заголовков столбцов
- Одно поле в качестве заголовков строк, два для заголовков столбцов и одно поле значений

115. Чтобы открыть схему данных, необходимо выполнить команду:

- Сервис, Схема данных
- Вставка, Схема данных
- Вставка, Таблица, Схема данных
- Правка, Вставить, Схема данных

Работа с современными промышленными СУБД/СУБД MS Access. Организация пользовательского интерфейса

116. На рисунке показан макет отчета по агентам, созданный в режиме

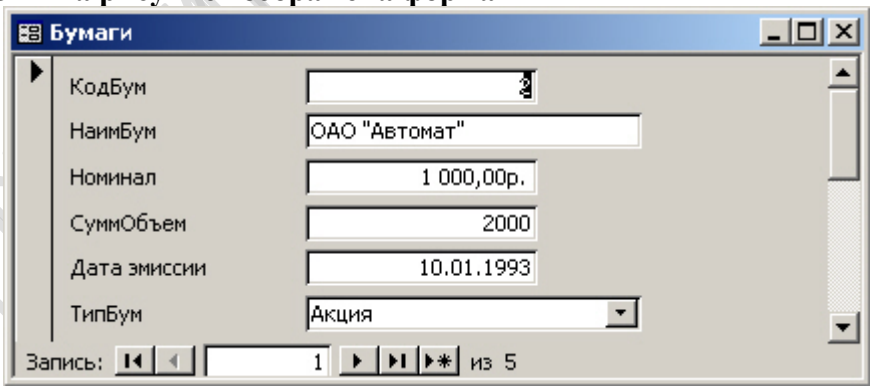


- Автоотчет ленточный
- Автоотчет в столбец
- Автоотчёт табличный
- Нет правильного ответа

117. Кнопка "Конструктор" открывает

- Структуру объекта
- Содержимое таблицы
- Панель элементов
- Выводит на печать таблицу

118. На рисунке изображена форма



- В столбец
- Ленточная
- Табличная
- Нет правильного ответа

119. Какого раздела не существует в конструкторе форм?

- ○ Заголовка
- ○ Верхнего колонтитула
- ○ Примечание
- ● Итоговый

120. При помощи какой панели устанавливаются кнопки в форме?

- ○ Стандартная
- ○ Кнопки
- ● Элементов
- ○ Форматирования

121. Из чего состоит макрос?

- ○ Из набора тегов
- ○ Из совокупности операторов Visual Basic
- ○ Из набора гиперссылок
- ● Из набора макрокоманд

122. Кнопочная форма Access создаётся:

- ● Конструктором форм
- ○ Мастером форм
- ○ Автоформами
- ○ Редактором форм

123. Переключатели в форме устанавливаются с помощью панели:

- ○ Стандартная
- ○ Формы
- ● Элементов
- ○ Переключателей

124. Мастер отчётов -

- ● Запускает основного мастера для создания отчетов, позволяющего выбрать поля для отчета, задать форматы, условия группировки и итоговые функции
- ○ Помогает построить диаграмму и создает в отчете свободную рамку объекта OLE с внедренной диаграммой Microsoft Graph
- ○ Позволяет создавать отчет «с нуля» и редактировать уже созданный отчет
- ○ Нет правильного ответа

125. В Microsoft Access 2000 содержится список макрокоманд, сгруппированных по категориям:

- Работа с данными в формах и отчетах
- Выполнение команд, макросов, процедур и запросов
- Работа с объектами
- Всё перечисленное

Функциональные зависимости и поиск данных/Функциональные зависимости на данные

126. В процессе чего следует избавиться от всех "других" функциональных зависимостей, т.е. от тех, которые имеют иной вид, чем K=>F (K - первичный ключ, а F - некоторое другое поле)?

- Составления схемы данных
- Нормализации
- Создания отчета
- Нет правильного ответа

127. Что такое представление?

- Файл с последовательностью пар ключей и указателей на запись в файле данных
- Виртуальная (логическая) таблица, представляющая собой поименованный запрос
- Объект базы данных, создаваемый с целью повышения производительности поиска данных
- Нет правильного ответа

128. Что такое индекс?

- Объект базы данных, создаваемый с целью повышения производительности поиска данных
- Подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости)
- Оба верны
- Нет правильного ответа

129. Что такое редкий индекс?

- Файл с последовательностью пар ключей и указателей
- Файл с последовательностью пар ключей и указателей на запись в файле данных
- Оба верны
- Нет правильного ответа

130. Какой первичный ключ называется простым?

- Первичный ключ состоит из трех атрибутов
- Первичный ключ состоит из двух атрибутов

- Первичный ключ состоит из единственного атрибута
- Нет правильного ответа

131. Какой первичный ключ называется составным?

- Первичный ключ состоит из трех атрибутов
- Первичный ключ состоит из двух и более атрибутов
- Первичный ключ состоит из единственного атрибута
- Нет правильного ответа

132. Выберите из предложенных вариантов аксиому аддитивности:

- Если X =>Y и Y=>Z, то X=>Z (можно выявлять неявные зависимости)
- Если X=>YZ, то X=>Y (можно отрезать атрибуты справа)
- Если X=>Y и X=>Z, то X=>YZ (можно объединять зависимости с одинаковыми левыми частями)
- Нет правильного ответа

133. Выберите из предложенных вариантов аксиому транзитивности:

- Если X =>Y и Y=>Z, то X=>Z (можно выявлять неявные зависимости)
- Если X=>YZ, то X=>Y (можно отрезать атрибуты справа)
- Если X=>Y и X=>Z, то X=>YZ (можно объединять зависимости с одинаковыми левыми частями)
- Нет правильного ответа

134. Какой граф считается ациклическим?

- Граф, содержащий один цикл
- Граф, не содержащий ни одного цикла
- Граф, содержащий два и более циклов
- Нет правильного ответа

135. Какие зависимости бывают?

- Однозначные
- Многозначные
- Оба верны
- Нет правильного ответа

Функциональные зависимости и поиск данных/Проблема аномалии и задача нормализации данных

136. Причинами аномалий при работе с данными являются:

- Хранение в одном отношении разнородной информации

- Схема отношения выполнена с соблюдением правил нормализации
- Избыточность данных, также порожденная тем, что в одном отношении хранится разнородная информация
- Каждый неключевой атрибут зависит от ключевого

137. Какие действия должны быть произведены для устранения всех типов аномалий (ввода,удаления,обновления)?

- Нормализация исходных схем
- Индексация схем
- Организация ссылочной целостности
- Закрытие прав доступа на работу с таблицей посторонним лицам

138. Нормальная форма — свойство отношения в реляционной модели данных, (…), которая потенциально может привести к логически ошибочным результатам выборки или изменения данных.

- Характеризующее его с точки зрения избыточности
- Характеризующее его с точки зрения надежности
- Характеризующее его с точки зрения упрощённости
- Характеризующее его с точки зрения аутентичности

139. Отношение находится в BCNF(нормальная форма Бойса-Кодда) тогда и только тогда, когда каждая ее нетривиальная и неприводимая слева функциональная зависимость имеет в качестве своего детерминанта:

- Сложный кортеж
- Потенциальный индекс
- Некоторый потенциальный ключ
- Строку состояния

140. Нормальная форма Бойса-Кодда представляет собой более строгую версию:

- Второй НФ
- Третьей НФ
- Первой НФ
- Четвертой НФ

141. Отношение находится в 3 НФ тогда и только тогда, когда выполняются следующие условия (отметьте два варианта):

- **50%** Отношение находится во второй нормальной форме
- **-100%** Отношение находится в первой нормальной форме (1НФ)
- **-100%** Таблица не содержит нетривиальных многозначных зависимостей
- **50%** Каждый неключевой атрибут отношения находится в нетранзитивной (то есть прямой) зависимости от потенциального ключа

142. Отношение находится в первой нормальной форме (1НФ) тогда и только тогда:

- ○ Когда таблица не содержит нетривиальных многозначных зависимостей
- ● Когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов
- ○ Когда в любом допустимом значении отношения каждый его кортеж может содержать несколько значений для каждого из атрибутов

143. Общее назначение процесса нормализации заключается в следующем:

- ○ Исключение некоторых типов избыточности
- ○ Устранение некоторых аномалий обновления
- ○ Упрощение процедуры применения необходимых ограничений целостности
- ○ Разработка проекта базы данных, интуитивно понятного и достаточно качественного, для возможности его дальнейшего расширения
- ● Верны все варианты ответов

144. Конечной целью нормализации является:

- ○ Уменьшение физического объёма БД
- ○ Увеличение физического объёма БД
- ● Уменьшение потенциальной противоречивости хранимой в БД информации
- ○ Увеличение производительности работы

145. Какой процесс называется нормализацией?

- ○ Процесс устранения внутренних аномалий БД
- ● Процесс преобразования отношений базы данных к виду, отвечающему нормальным формам
- ○ Процесс преобразования базы данных к виду, наиболее близкому к НФ Бойса-Кодда
- ○ Любой процесс увеличивающий избыточность данных

Функциональные зависимости и поиск данных/Использование функциональных зависимостей при работе с данными

146. В каких случаях применяется функция хэширования (отметьте три варианта):

- ○ **-100%** Удаление данных
- ● **33%** Проверка на наличие ошибок
- ○ **-100%** Объединение данных в массивы и последующая сортировка
- ● **33%** Сверка данных
- ● **34%** Ускорение поиска данных

147. Какая сортировка чаще всего используется в СУБД:

- Топологическая сортировка
- Блинная сортировка
- Внешняя сортировка
- Сортировка Гаппо

148. Основная характеристика вычислительной сложности алгоритма:

- Время
- Объем алгоритма
- Наличие вложенных циклов
- Количество требуемой памяти

149. Укажите утверждение не характерное для ассоциативного поиска в массиве:

- Для ускорения операции поиска можно упорядочить элементы массива (ассоциативный массив) по ключу и осуществлять поиск методом деления пополам
- Для поиска в ассоциативном массиве невозможно использовать хэш-таблицы
- Использование деревьев поиска в ассоциативном массиве

150. Так как физические записи имеют разную длину, то при модификации данных запись может увеличиться и превысит исходную длину записи до модификации. В этом случае при определенных методах хранения может понадобиться дополнительное пространство хранения, где и будут размещены дополнительные данные. Это пространство называется:

- Добавочное пространство
- Область переполнения
- Внешняя область
- Внутренняя область

151. ::Оператор:: Какой оператор в SQL определяет, совпадает ли указанная символьная строка с заданным шаблоном?

- like

152. Какой комбинацией клавиш в MS Access вызывается окно поиска?

- CTRL+F
- CTRL+U
- CTRL+ALT+W
- CTRL+H

153. ::Ключевое слово:: Напишите, какое слово необходимо использовать для сортировки по убыванию:
SELECT DISTINCT Predmety.Predm
FROM Predmety
ORDER BY Predmety.Predm [СЛОВО];

- desc

154. Может ли сортировка в MS Access выполняться по нескольким столбцам?

- Затрудняюсь ответить
- Нет
- Да

Функциональные зависимости и поиск данных/Языки баз данных

155. Выберите правильное понятие языка манипулирования данными:

- Первый язык программирования высокого уровня, имеющий транслятор
- Язык разметки гипертекста
- Семейство компьютерных языков, используемых в компьютерных программах или пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных
- Язык и система программирования, предназначенная для поддержки начальных курсов информатики и программирования в средней и высшей школе

156. Укажите наиболее популярный на текущий момент язык DML:

- ЛИНТЕР
- SQL
- MongoDB
- CouchDB

157. Укажите преимущество SQL:

- Лёгкая возможность перенести тексты SQL-запросов из одной СУБД в другую
- Отсутствие поддержки свойства «=»
- Неопределённые значения (nulls)
- Повторяющиеся строки

158. Укажите год первого варианта стандарта SQL, принятый институтом ANSI:

- 1986
- 1989
- 1992
- 1999

159. Выберите объявления, относящиеся к точным числам (отметьте два варианта):

- **-100%**VARCHAR
- **50%**DECIMAL

- **50%**SMALLINT
- **-100%**INTERVAL

160. Что определяет тип BLOB?

- Набор допустимых значений для одного или нескольких атрибутов
- Может хранить не более 65 535 символов
- Дробное число, хранящееся в виде строки
- Число с плавающей точкой двойной точности

161. Какой запрос используется для получения информации, хранящейся в базе данных?

- DELETE
- UPDATE
- INSERT
- SELECT

162. Какой запрос используется для создания новой строки данных?

- UPDATE
- INSERT
- DELETE
- SELECT

163. Используя какой запрос можно удалить таблицы (TABLE), индексы (INDEX) и базы данных (DATABASE)?

- DROP
- UPDATE
- REVOKE
- ALTER TABLE

164. При формировании запроса SELECT пользователь описывает ожидаемый набор данных:

- Тип и значения данных
- Вид и содержимое
- Вид
- Содержимое

Функциональные зависимости и поиск данных/Использование языка запросов SQL для выборки данных

165. Дайте определение оператору SELECT

- Оператор DML языка SQL, возвращающий набор данных (выборку) из базы данных, удовлетворяющих заданному условию
- Оператор в SQL, указывающий, что оператор языка управления данными (DML) должен действовать только на записи, удовлетворяющие определенным критериям
- Оба варианта возможны
- Нет верного определения

166. Какие ключевые слова относятся к запросу SELECT?

- WHERE , HAVING
- GROUP BY
- ORDER BY
- Все варианты верны

167. SELECT — оператор, возвращающий набор данных из базы данных, удовлетворяющих заданному условию. В большинстве случаев, выборка осуществляется:

- Из одной таблицы
- Из нескольких таблиц
- Из одной или нескольких таблиц
- Все варианты неверны

168. Формат запроса с использованием оператора SELECT

- SELECT список полей FROM список таблиц WHERE условия…
- SELECT список полей GROUP BY список таблиц WHERE условия…
- SELECT список полей FROM список таблиц ORDER BY условия…
- Нет верного ответа

169. Следующее утверждение верно:

- Алиас - псевдоним для конкретной таблицы в некоторой БД
- Алиас - имя для абстрактной таблицы в некоторой БД
- Алиас - имя для таблицы в некоторой БД
- Все варианты неверны

170. При формировании запроса SELECT пользователь описывает ожидаемый набор данных:

- Тип и значения данных
- Вид и содержимое
- Вид
- Содержимое

171. Управление полями — это

- Указание полей таблицы (таблиц), исключаемые из результирующего набора данных
- Указание полей таблицы (таблиц), включаемых в компенсирующий набор данных
- Указание полей таблицы (таблиц), включаемых в результирующий набор данных
- Нет верных ответов

172. SELECT используется:

- Для отбора записей, удовлетворяющих сложным критериям поиска
- Для вывода, удовлетворяющих сложным критериям поиска
- Все варианты верны
- Все варианты неверны

173. Результатом выполнения оператора SELECT является:

- Действие
- Таблица
- Действие и таблица
- Все ответы неверны

174. Как выглядят с точки зрения оператора SELECT постоянно хранимые таблицы и временные таблицы?

- Одинаково
- Различно
- Различия скрыты от пользователя, но при реальном выполнении учитываются
- Все ответы неверны

Функциональные зависимости и поиск данных/Отбор и сортировка записей в запросах для выборки данных

175. При помощи какого ключевого слова в операторе Select можно исключить повторяющиеся записи?

- Join
- Having
- Distinct
- Like

176. Каким образом происходит выборка значений из нескольких таблиц?

- Нужно провести суммирование таблиц и далее провести запрос по получившейся таблице
- Достаточно указать поля, по которым идет отбор, а таблицу запрос определит сам
- Нужно указать таблицы в списке таблиц

- Нужно перед выборкой провести операцию реляционной алгебры проекция и по получившемуся множеству записей провести запрос

177. Какие операции можно применять в предложении Where?

- Любые
- Логические
- Математические
- Операции реляционной алгебры

178. Определение предложения Group by:

- Данное предложение позволяет определить множество значений отдельного поля в терминах другого поля и применить к ним функции агрегирования
- Данное предложение позволяет генерировать значения истина, если хоть одно из группируемых значений выдает истину
- Данное предложение позволяет генерировать значения ложь, если хоть одно из группируемых значений выдает ложь
- Данное предложение позволяет определить набор записей, к которым будет применятся функция агрегирования

179. Каково назначение оператора Like?

- Создает набор из допустимых числовых значений
- Является аналогом логической функции OR
- Является аналогом логической функции AND
- Задает текстовую строку с возможным использованием метасимволов

180. Каково назначение оператора Is null?

- Обнуляет таблицу, задавая всем ячейкам значения null
- Служит для сравнения с ячейками, значения которых null
- Служит для поиска значений null в таблице
- При указании этого оператора все столбцы и строки содержащие значения null удаляются

181. Аналогом какого логического выражения является оператор In?

- имя_поля = значение 1 or имя_поля = значение 2
- имя_поля >= значение 1 and имя_поля <= значение 2
- имя_поля like «шаблон»
- имя_поля not логического значение

182. Выберите правильное выражение (отметьте два варианта):

- **50%** Asc – сортировка по возрастанию
- **50%** Desc – сортировка по убыванию

- -100%Asc – сортировка по убыванию
- -100%Any – сортировка по убыванию

183. Аналогом какого логического выражения является оператор Between?

- имя_поля = значение 1 or имя_поля = значение 2
- имя_поля >= значение 1 and имя_поля <= значение 2
- имя_поля like «шаблон»
- имя_поля not логического значение

184. Для какой цели применяются функции агрегирования?

- Для вычисления операции разность реляционной алгебры
- Для составления подзапросов
- Для расчета значений по группам строк
- Для работы с индексами таблиц

Функциональные зависимости и поиск данных/Группировка данных в запросе

185. Процедура объединения в логическом порядке строк с определенными значениями это:

- Сортировка данных
- Группировка данных
- Суммирование данных
- Обобщение данных

186. Одним из видов группировки является сортировка (определяемая фразой …), кроме этого в группе условий оператора выборки SELECT могут быть использованы фразы группировки … и … . Выберите вариант ответа, в котором записаны фразы в порядке их замены троеточиями.

- ORDER BY, GROUP BY, HAVING
- HAVING, WHERE, COUNT
- ORDER BY, AVG, HAVING
- GROUP BY, ORDER BY, SUM

187. Фраза GROUP BY с синтаксисом: … позвляет преобразовать таблицу так, что в ее отдельные строки будет собрано содержание всех строк с одинаковыми значениями полей группировки. Выберите вариант ответа с правильным синтаксисом.

- GROUP BY SELECT имя_поля [,имя_поля ...]
- GROUP BY FROM имя_поля [,имя_поля ...]
- GROUP BY имя_таблицы [,имя_таблицы ...]
- GROUP BY имя_поля [,имя_поля ...]

188. Для каких полей используется группировка ?

- Используется для тех полей, по значениям которых нужно сгруппировать записи таблицы
- Используется для тех полей, значения которых не надо группировать
- Используется для тех полей, по значениям которых нужно удалить записи таблицы
- Используется для пустых полей

189. Функции COUNT, MAX, MIN, SUM, AVG предназначенные для расчета значений по группам строк таблиц называются:

- Функции группировки
- Функции множеств
- Функции агрегирования
- Функции объединения

190. Выберите варианты ответа с правильными запросом (отметьте два ответа).

- **50%**SELECT COUNT(stock) FROM stock WHERE stock <> 0
- **-100%**SELECT * FROM tab1 WHERE col1 = MAX(col1)
- **50%**SELECT AVG(OCEN) FROM ОЦЕНКА WHERE SNUM=”003”
- **-100%**SELECT SUM(FAMILIA) FROM FIO WHERE SNUM=АБВ

191. Из функций агрегирования только с числовыми полями могут работать?

- COUNT, MAX
- MAX, MIN, SUM, AVG
- SUM, AVG
- SUM

192. Из функций агрегирования работать с числовыми и символьными полями могут?

- MAX, MIN, SUM, AVG
- MAX, SUM, AVG
- MAX, MIN
- COUNT, MAX, MIN

193. Выберите варианты ответов с корректными запросами (отметьте два ответа):

- **50%**SELECT OSNUM, AVG(OCEN) FROM OCENKA HAVING AVG(OCEN)>=4
- **-100%**SELECT OSNUM, AVG(OCEN) FROM OCENKA WHERE AVG(OCEN)>=4 GROUP BY OSNUM
- **-100%**SELECT OSNUM, MIN(OCEN) FROM OCENKA GROUP BY OSNUM HAVING ODATE > 14/01/10

- **50%**SELECT OSNUM, MIN(OCEN) FROM OCENKA GROUP BY OSNUM WHERE ODATE > 14/01/10

194. Какой аргумент группировки должен иметь единственное значение для каждой выходной группы?

- COUNT
- HAVING
- SELECT
- GROUP BY

Функциональные зависимости и поиск данных/Выборка данных из множества таблиц

195. Выберите верное утверждение:

- Самообъединение – объединение таблицы с этой же таблицей – является другим вариантом объединения таблиц на основе операции эквисоединения. В этом случае сравниваются значения внутри столбца одной таблицы
- Самообъединение объединяет вывод двух или более SQL запросов в единый набор строк и столбцов
- Самообъединение позволяет определять множество значений отдельного поля в терминах другого поля
- Самообъединение позволяет вкладывать запросы друг в друга

196. Укажите пример без декартового произведения:

- SELECT Меню.*, Трапезы.*, Вид_блюд.*, Блюда.*
  FROM Меню, Трапезы, Вид_блюд, Блюда;
- SELECT Меню.*, Трапезы.*, Вид_блюд.*, Блюда.*
  FROM Меню, Трапезы, Вид_блюд, Блюда
  WHERE Меню.Т = Трапезы.Т
  AND Меню.В = Вид_блюд.В
  AND Меню.БЛ = Блюда.БЛ;
- SELECT Вид_блюд.*, Трапезы.*
  FROM Вид_блюд, Трапезы;
- SELECT блюда
  FROM Вид_блюд WHERE Вид_блюд = салаты

197. Операция Left Join:

- Объединяет записи двух таблиц при выполнении условий
- Создает левое внешнее объединение, при котором все записи из первой таблицы включаются в результат, даже если со второй правой таблицы нет соответствия по условию записей

- Создает правое внешнее объединение, при котором все записи из второй таблицы включаются в результат, даже если с первой левой таблицы нет соответствия по условию записей
- Нет правильного варианта

198. Самым распространенным типом объединения можно считать:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- Все они одинаково распространены

199. Выберите верные утверждения (отметьте два варианта):

- -100%Операция INNER JOIN может включаться в операцию LEFT JOIN
- 50%Операция LEFT JOIN может включаться в операцию INNER JOIN
- 50%Операция RIGHT JOIN может включаться в операцию INNER JOIN
- -100%Операция INNER JOIN может включаться в операцию RIGHT JOIN

200. Выберите связующее поле:
SELECT ИмяКатегории, Наименование, Время, Сложность
FROM Категории INNER JOIN Товары
ON Категории.ИДКатегории = Товары.ИДКатегории;

- ИДКатегории
- Наименование
- Время
- Сложность

201. Выберите два возможных оператора:
…FROM таблица1 INNER JOIN таблица2 ON таблица1.поле1 ВОЗМОЖНЫЙ ОПЕРАТОР таблица2.поле2…

- 50%=
- -100%AND
- -100%OR
- 50% <>

202. SELECT Вид_блюд.*, Трапезы.*
FROM Вид_блюд, Трапезы;
В результате этого запроса, если видов блюд 5, а трапез 3, мы получим таблицу, содержащую:

- 8 строк
- 15 строк
- 2 строки

- 5 строк

203. **SELECT sname, pname**
FROM преподаватель, студент
WHERE pnum=spdp
GROUP BY sname DESC;
Это SQL запрос на выборку данных из множества таблиц. Результатом этого запроса будут:

- Все строки, входящие в таблицы: преподаватель, студент
- Строки, входящие в таблицу преподаватель и не входящие в таблицу студент
- Строки таблиц преподаватель и студент, в которых поля pnum и spdp не совпадают
- Строки таблиц преподаватель и студент, в которых поля pnum и spdp совпадают

204. Менее ресурсоемким типом объединения можно считать:

- CROSS JOIN
- LEFT JOIN
- FULL JOIN
- Все варианты одинаково ресурсоемки

Функциональные зависимости и поиск данных/Использование подзапросов в языке запросов SQL

205. Какой из команд использующей подзапросы можно изменить содержание строк в базе данных?

- DELETE
- UPDATE
- INSERT
- SELECT

206. Как включаются подзапросы в приложении WHERE? (отметьте три варианта)

- **33%** С помощью условия IN
- **33%** С помощью условия EXISTS
- **-100%** Записываются через запятую
- **34%** С помощью условий сравнения (= | <> | < | <= | > | >=)

207. Определите порядок проведения процедура оценки, выполняемой связанным запросом:
1.Выполнить подзапрос. Для отбора записей использовать строку-кандидат.
2.Выбрать строку из таблицы, указанной во внешнем запросе. Это будет текущая строка-кандидат.
3.Вычислить условие во внешнем запросе, на основе результатов внутреннего подзапроса, выполняемого в п.3. Определяется - отбирается ли строка-кандидат

для вывода.
4.Повторить процедуру для всех строк.
5.Сохранить значения из этой строки-кандидата во временном буфере.

- 1, 2, 3, 4, 5
- 2, 4, 1, 3, 5
- 2, 5, 1, 3, 4
- 3, 2, 1, 4, 5

208. Из приведенный ниже подзапросов выберите связанный запрос.

- SELECT COUNT(*) AS orders
  FROM Orders
  WHERE cust_id = '1000000001';
- SELECT *
  FROM Customers C
  WHERE '1999-10-03' IN (
  SELECT odate
  FROM Orders O
  WHERE O.cnum = C.cnum)
- SELECT cust_name, cust_state,(SELECT COUNT(*)
  FROM Orders
  WHERE Orders.cust_id = Customers.cust_id) AS orders FROM Customers ORDER BY cust_name;
- SELECT *
  FROM Orders outer
  WHERE amt >
  (SELECT AVG amt
  FROM Orders inter
  WHERE inner.cnum = outer.cnum);

209. Как происходит связывание таблицы со своей копией?

- При помощи выражения WHERE
- При помощи выражения HAVING
- Командой SELECT
- Командой FROM

210. Какой из операторов может вывести NULL-значения?

- ANY
- ALL
- EXISTS
- SOME
- NOT EXISTS

211. Оператор EXISTS используется:

- ○ Чтобы подзапрос не выводился
- ○ Чтобы произвести вывод подзапроса
- ● Чтобы указать предикату, производить ли подзапросу вывод или нет
- ○ Для всего перечисленного

212. Чем отличается ANY от EXISTS? (отметьте два варианта)

- ○ **-100%** Подзапрос не выбирает значения такого же типа как и те, которые сравниваются в основном предикате
- ● **50%** Подзапрос должен выбирать значения такого же типа как и те, которые сравниваются в основном предикате
- ● **50%** Нет возможности работать со значениями NULL
- ○ **-100%** Есть возможность работать со значениями NULL

213. Операторы SOME и ANY:

- ● Взаимозаменяемы везде
- ○ Не взаимозаменяемы
- ○ Использоваться только с подзапросами
- ○ Все варианты не верны

214. С помощью ALL предикат будет верным?

- ● Если каждое значение, выбранное подзапросом, удовлетворяет условию в предикате внешнего запроса
- ○ Если каждое значение, выбранное подзапросом, не удовлетворяет условию в предикате внешнего запроса
- ○ Если каждое значение, выбранное подзапросом, удовлетворяет условию в предикате внутреннего запроса
- ○ Если каждое значение, выбранное подзапросом, не удовлетворяет условию в предикате внутреннего запроса

++

Функциональные зависимости и поиск данных/Объединение запросов

215. Какое из перечисленных ниже предложений является объединением?

- ○ HEAVING
- ● UNION
- ○ GROUP BY
- ○ WHERE
- ○ UPDATE

216. Какую роль выполняет предложение UNION?

- ○ Определяет критерий включения записей по группам в результат
- ○ Сортирует записи, возвращенные запросом, по возрастанию или по убыванию значений указанного поля (полей)
- ○ Группирует указанному перечню столбцов с тем, чтобы получить для каждой группы единственное агрегированное значение
- ○ Выбирает данные из указанных столбцов и (если необходимо) выполнить перед выводом их преобразование в соответствии с указанными выражениями и (или) функциями
- ● Позволяет получить отношение, состоящее из всех строк, входящих в одно или оба объединяемых отношения

217. Какие существуют условия совместимости запросов для объединения? (отметьте два варианта)?

- ● **50%** Каждый запрос должен указывать одинаковое число столбцов
- ○ **-100%** Каждый запрос должен указывать одинаковое число строк
- ● **50%** Когда пустые значения(NULL) запрещены в любом столбце объединения
- ○ **-100%** Запросы должны иметь одинаковое число подзапросов

218. Какое количество запросов позволяет объединять UNION?

- ○ 2 запроса
- ○ Не более 3-х
- ○ До 13 запросов
- ● Любое число запросов

219. В чём отличие предложения Union от подзапросов?

- ● В нем ни один из двух (или больше) запросов не управляются другим запросом
- ○ В нем запросы управляются одним из запросов запросов
- ○ Ничем

220. Как можно задать объединение таблиц?

- ● При помощи предложения UNION
- ○ При помощи JOIN
- ○ При помощи ORDER BY

221. Что такое Union?

- ● Предложение
- ○ Запрос
- ○ Выражение
- ○ Таблица

- Подзапрос

222. Получить повторяющиеся строки в МА можно с помощью:

- Union ALL
- NOT NULL
- Union NOT NULL

223. Выберете неправильное выражение:

- SELECT osnum, MIN(ocen)
  FROM оценка GROUP BY osnum
  UNION
  SELECT osnum, MAX(ocen)
  FROM оценка GROUP BY osnum;
- SELECT pnum, pname FROM преподаватель
  WHERE pnum IN
  (SELECT DISTINCT spdp FROM студент)
  UNION
  SELECT snum, sname FROM студент
  WHERE spdp IS NOT NULL;
- SELECT pnum
  UNION преподаватель;

224. Для описания множеств, получающихся при пересечении и объединении таблиц, какой используется специальный математический аппарат?

- Реляционная алгебра
- Аналитическая геометрия
- Математический анализ

Функциональные зависимости и поиск данных/Запросы обновления таблиц в языке запросов SQL

225. Выберите правильный формат команды обновления данных:

- INSERT INTO список_значений [(имя_таблицы)] VALUES (список_имен)
- UPDATE имя_таблицы SET поле=значение [, поле=значение, …] WHERE условие
- DELETE FROM имя_таблицы VALUES (список_значений)

226. Выберите утверждение, справедливое при работе с командой INSERT:

- Согласно стандарту SQL, допускается, что вводимые значения могут быть выражениями
- При указании списка имен полей их порядок должен совпадать с порядком полей в таблице
- Список значений должен указываться в порядке списка имен полей

227. Выберите неверный запрос:

- INSERT INTO A SELECT a FROM B WHERE a=2
- INSERT INTO A SELECT * FROM A WHERE a=2
- INSERT INTO A SELECT b FROM B WHERE a=2

228. Выберите правильный запрос на обновление поля a в таблице A на значение t при условии x:

- UPDATE A SET x=t WHERE a
- UPDATE a SET a=t WHERE x
- UPDATE t SET A=a WHERE x
- UPDATE A SET a=t WHERE x

229. Выберите правильный запрос на обновление записей, где a, b – поля, A, B – таблицы, x - условие:

- UPDATE A SET a=1 WHERE a=(SELECT b FROM B WHERE x)
- UPDATE A SET a=1 WHERE (SELECT b FROM B WHERE x)=a
- UPDATE A SET a=1 WHERE a=(SELECT b FROM A WHERE x)

230. Выберите правильный запрос на удаление из таблицы A всех записей:

- DELETE FROM A
- DELETE FROM A WHERE NULL
- DELETE FROM A WHERE *

231. Как правильно удалять информацию из взаимосвязанных таблиц?

- Удалить информацию из главной таблицы, затем из подчиненных
- Удалить информацию из подчиненных таблиц, затем из главной
- Удалить информацию из главной таблицы

232. Где в команде DELETE могут использоваться подзапросы?

- При выборе полей таблицы
- В команде DELETE подзапросы не используются
- В условии отбора удаляемых записей

233. Где в команде UPDATE могут использоваться подзапросы:

- При установке нового значения поля
- В условии отбора обновляемых записей
- При выборе обновляемых таблиц

234. Что может использоваться в качестве значений полей при работе с командой UPDATE (отметьте два варианта)?

- **50%** Константы

- -100%Переменные
- 50%Выражения от значений полей текущей записи

Функциональные зависимости и поиск данных/Оптимизация выполнения запросов SQL

235. Метод оптимизации выполнения запросов, основанный на синтаксисе:

- При использовании этого метода оптимизатор не рандомно выбирает оптимальный план выполнения запроса
- При использовании этого метода оптимизатор начинает начинает работу без проверки оптимального плана выполнения запроса
- При использовании этого метода план составляется на основании существующих путей доступа и их рангов. Все пути доступа ранжируются на основании знаний о правилах и последовательности осуществления этих путей
- Нет правильного ответа

236. Если оптимизатор основывается только на информации о механизмах реализации путей доступа, то метод оптимизации основан на:

- Основан на стоимости
- Основан на синтаксисе
- Основан на синтаксисе и стоимости
- Нет правил иного ответа

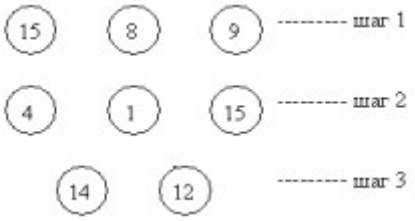
237. Укажите ранги путей доступа для СУБД основанный на синтаксисе (отметьте три варианта):

- 33%MAX и MIN по индексированному столбцу
- 33%ORDER BY по индексированному столбцу
- 34%Полный просмотр таблицы
- -100%Просмотр таблицы с конца
- -100%Просмотр таблицы с середины

238. Укажите ранги путей доступа для СУБД основанный на синтаксисе (отметьте три варианта):

- 33%Ключ индексного кластера
- 33%Составной индекс
- 34%ORDER BY по индексированному столбцу
- -100%Неполный просмотр таблицы
- -100%Нет правильного ответа

239. Постройте план выполнения запроса реализованном методом оптимизации по стоимости:



- 15, 4, 14
- 8, 1, 12
- 9, 15, 12
- 8, 1, 14
- Нет правильного варианта

240. Укажите номер в котором описывается метод оптимизации, основанный на стоимости:

- При использовании этого метода оптимизатор сначала строит несколько возможных планов выполнения запроса. Для выбора наиболее перспективных планов оптимизатор в самом начале отбрасывает неэффективные планы и не рассматривает их
- При использовании этого метода оптимизатор строит один возможный план выполнения запроса
- При использовании этого метода оптимизатор строит несколько возможных планов, для выбора более перспективных планов оптимизатор в самом начале отбрасывает эфективные планы
- Нет правильного ответа

241. Если оптимизатор основывается только на информации о механизмах реализации путей доступа , и помимо этого используется статистическая информация о распределении данных, то это метод оптимизации, основанный на:

- Основан на синтаксисе и стоимости
- Основан на стоимости
- Основан на синтаксисе
- Нет правильного ответа

242. Расположите поэтапно оптимизацию запросов в реляционных СУБД:
1)Выбор плана выполнения запроса
2)лексический и синтаксический анализ
3)Выполнение плана
4)Логическая и семантическая оптимизация
5)Генерация процедурного плана выполнения запроса

- 1,2,3,4,5
- 2,4,1,5,3
- 1,3,4,2,5

- 5,3,1,2,4
- 2,4,3,1,5,

243. Укажите к какой фазе, оптимизации выполнения запроса, относятся следующие действия. Лексический анализатор разбивает запрос на лексические единицы – лексемы (наименования полей и таблиц, константы, знаки операций и т.д.). Синтаксический анализатор проверяет синтаксическую правильность запроса.

- Лексический и синтаксический анализ
- Выполнение плана
- Генерация процедурного плана выполнения запроса
- Выбор плана выполнения запроса
- Логическая и семантическая оптимизация

244. Для чего нужно использовать Оптимизация SQL запросов?

- Для улучшения читаемости кода базы данных
- для улучшения интерфейса базы данных
- Для увеличения скорости базы данных
- Нет правильного ответа

Проектирование баз данных/Этапы проектирования баз данных

245. В результате инфологического проектирования БД должна быть создана:

- Инфологическая модель
- Схема БД
- Концептуальная модель
- Реляционная модель данных

246. Концептуальная модель (инфологическая модель) базы данных включает в себя (отметьте два варианта):

- **-100%** Набор схем отношений
- **50%** Описание информационных объектов
- **50%** Описание ограничений целостности
- **-100%** Описание внешних ключей

247. Отметьте 2 преимущества ER моделей:

- **50%** Модели позволяют проектировать базы данных с большим коли- чеством объектов и атрибутов
- **-100%** Атрибуты объектов
- **-100%** Связь

- 50% Наглядность

248. Выберите 2 элемента ER-моделей:

- 50% Связи между объектами
- 50% Атрибуты объектов
- -100% Связь
- -100% Набор схем отношений

249. Связь между сущностями характеризуется (отметьте два варианта):

- -100% Внешними ключами
- 50% Типом связи (1:1, 1:N, N:M)
- 50% Классом принадлежности
- -100% Атрибутами ключей

250. Структура запросов в предметном подходе к проектированию (отметьте два варианта):

- 50% Не определена
- 50% Определена но не полностью
- -100% Определена
- -100% Определена связями

251. Предметный подход к проектированию БД применяется в тех случаях когда (отметьте два варианта):

- 50% Есть представление какую именно информацию они хотели бы хранить в БД
- -100% Нет возможности использования подхода другого вида
- -100% У разработчиков нет чёткого представление о самой ПО
- 50% У разработчиков есть чёткое представление о самой ПО

252. Основное внимание в предметном подходе к проектированию уделяется:

- Составлению схемы данных
- Исследованию ПО
- Проектированию ПО
- Проектированию БД

253. Функциональный подход применяется когда известны (отметьте два варианта):

- 50% Функции некоторой группы лиц
- -100% Схемы БД

- 50%Функции комплекса задач
- -100%Данные для заполнения базы

254. Функциональный подход реализует принцип:

- Проектирования
- "Сущность–связь"
- Логического проектирования БД
- "От задач"

Проектирование баз данных/Элементы проектирования баз данных

255. От чего зависит объем вычислительных ресурсов?

- От предполагаемого объема проектируемой базы данных
- От интенсивности использования базы данных
- От использования в многопользовательском или однопользовательском режиме
- Все варианты правильные
- Все варианты неправильные

256. Какие критерии являются важнейшими при выборе СУБД (отметьте три варианта):

- 33%Удобство и надежность СУБД в эксплуатации
- 33%Стоимость СУБД и дополнительного программного обеспечения
- -100%Наличие графического интерфейса
- 34%Характеристики производительности системы
- -100%Характеристики производительности видео- и аудиоадаптера

257. Какой выбор принципиально влияет на весь процесс проектирования БД?

- Выбор ядра СУБД
- Выбор интерфейса
- Выбор средств резервного хранения данных БД
- Выбор средств защиты БД

258. От чего зависит логическое проектирование БД?

- От метода доступа к компонентам БД
- От производительности системы
- От модели данных, поддерживаемой данной СУБД
- От средств защиты СУБД

259. Что является результатом логического проектирования БД?

- Концептуальная схема БД
- Отображение структуры хранения БД
- Модель данных БД
- Реализация методов доступа к данным

260. Этап разработки БД, при котором происходит увязка логической структуры БД и физической среды, называется:

- Разработкой структуры данных
- Логическим проектированием
- Физическим проектированием
- Размещением данных
- Привязка базы

261. На какие аспекты в первую очередь необходимо обращать внимание при разработке защиты данных в СУБД (отметьте два варианта)?

- **50%** Защита от сбоев
- **-100%** Защита от неправильного ввода данных
- **50%** Защита от несанкционированного доступа
- **-100%** Защита от редактирования

262. Какая стратегия применяется для защиты данных от сбоев?

- Система паролей
- Резервное копирование
- Защита от редактирования
- Ограничение доступа к данным

263. Как называются средства автоматизации проектирования?

- FPR-средства
- CASE-средства
- AVPRO-средства
- MS-LOGIC

264. Как дословно переводится аббревиатура CASE-средств для автоматизации проектирования БД?

- Классическая автоматизация сервиса
- Разработка программного обеспечения с помощью компьютера
- Компьютерная автоматическая структура
- Структурная схема инженерии

Проектирование баз данных/Проектирование таблиц

265. Строка в отношении называется:

- ○ Атрибут
- ○ Домен
- ○ Мощность отношения
- ● Кортеж

266. Арностью отношения в реляционной терминологии определяется:

- ● Числом атрибутов
- ○ Числом кортежей
- ○ Числом доменов
- ○ Числом отношений

267. Какой вид взаимосвязей недопустим в реляционных базах данных:

- ○ Один-к-одному
- ○ Один-к-многим
- ● Многие-ко-многим
- ○ Все ответы являются допустимыми

268. При каком из видов взаимосвязи таблицы могут быть без потери сущности объединены?

- ● Один-к-одному
- ○ Один-к-многим
- ○ Многие-ко-многим
- ○ Нет правильного ответа

269. Отношение А, содержащее внешний ключ, связывающий его с другим отношением В, является:

- ○ Дочерним для всех отношений БД
- ○ Родительским для отношения В
- ○ Родительским для всех отношений БД
- ● Дочерним для отношения В

270. Таблица В связью вида «многие-к-одному» соотносится с таблицей А. Какие из указанных утверждений являются правильными для этого случая (отметьте два варианта):

- ● **50%** Таблица А является родительской
- ○ **-100%** Таблица В является родительской
- ○ **-100%** Таблица А является дочерней

- **50%** Таблица В является дочерней

271. Создание дополнительных отношений для удаления связей «многие-ко-многим» используется из-за того, что:

- Дополнительное отношение в любом случае приводит связь к виду «один-к-многим»
- Дополнительное отношение сводит множество таблиц к одной единственной
- Дополнительное отношение хранит идентификаторы связанных объектов (таблиц)
- Дополнительное отношение изменяет ключи связанных объектов в соответствии с реляционной моделью БД

272. Ключи дополнительного отношения, поддерживающие связанность и целостность базы данных, по отношению к ключам первоначальных таблиц являются:

- Первичными
- Составными
- Внешними
- Альтернативными

273. Количество ключевых атрибутов в таблице не должно превышать:

- 3
- Количество атрибутов не ограничено
- 30% от числа атрибутов отношения
- 50% от числа атрибутов отношения

274. В состав атрибутов могут быть включены (отметьте три варианта):

- **33%** Первичные ключи
- **33%** Внешние ключи
- **-100%** Кортежи доменов
- **34%** Неключевые значения доменов

Учреждение образования
«Гомельский государственный университет имени Франциска Скорины»

УТВЕРЖДАЮ
Ректор
ГГУ имени Ф. Скорины

_____ С.А. Хахомов
_____
(дата утверждения)
Регистрационный № УД-_____/
уч.

Модуль «Обработка данных»:
БАЗЫ И БАНКИ ДАННЫХ
Учебная программа учреждения высшего образования по учебной дисциплине для специальности
1-53 01 02 Автоматизированные системы обработки информации

2022 г.

Учебная программа составлена на основе образовательного стандарта ОСВО 1-53 01 02-2021 г. и учебного плана ГГУ имени Ф.Скорины регистрационный № I 53-1-21/УП, дата утверждения 31.05.2021.

СОСТАВИТЕЛЬ:
В.Н. Леванцов, старший преподаватель кафедры АСОИ

Кафедрой автоматизированных систем обработки информации
(протокол № 9 от 19.04.2022);

Научно-методическим советом учреждения образования «ГГУ имени Ф.Скорины»
(протокол № 4 от 17.05.2022).

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Изучение дисциплины компонента учреждения высшего образования «Базы и банки данных» предусмотрено учебным планом подготовки специалистов специальности 1-53 01 02 – «Автоматизированные системы обработки информации».

Целью изучения дисциплины является подготовка студентов по основам работы с базами данных в области управления и обработки информации.

Основные задачи изучения дисциплины:

-изучение технологий создания и работы с базами данных в современных системах управления базами данных (СУБД) и языках программирования;

- изучение технологий для работы с базами данных в сети;

- приобретение навыков по разработке программного интерфейса с современными СУБД.

Для изучения дисциплины «Базы и банки данных» необходимы базовые знания по следующим дисциплинам: «Основы алгоритмизации и программирования», «Основы информационных технологий».

В результате изучения дисциплины обучаемый студент **должен знать:**

- основные принципы организации баз данных;
- язык SQL;
- способы работы с реляционными базами данных в современных языках программирования;
- операции над таблицами;
- основы клиент-серверной технологии

уметь:

- применять современные технологии при разработке практических баз данных в автоматизированных информационных системах;
- создавать клиент-серверные системы;
- создавать визуальный интерфейс с базами данных;
- интегрировать работу с базами данных в программные приложения.

Дневная форма обучения (2 и 3 курс): Общее количество часов – 320; аудиторное количество часов — 140, из них 4 семестр: лекции — 26, лабораторных занятий - 18 часов, управляемая самостоятельная работа — 8. Форма отчётности – зачет.

5 семестр: лекции — 20, лабораторных занятий - 58 часа, управляемая самостоятельная работа — 10. Курсовое проектирование – 30 часов. Форма отчётности — экзамен.

Заочная форма обучения (2 и 3 курс): Общее количество часов – 320, аудиторное количество часов – 36, из них 4 семестр: лекционных занятий – 10, лабораторных занятий – 8. Форма отчётности – зачет.

5 семестр: лекционных занятий – 10, лабораторных занятий – 8, курсовое проектирование – 30 часов.

Заочная форма обучения дистанционная (интегрированная на основе среднего специального образования) 3 и 4 курс: количество часов – _320_ аудиторное количество часов – 20, из них 5 семестр: лекционных занятий – 2,

6 семестр: лекционных занятий – 4, лабораторных занятий – 6. Форма отчётности – зачет.

7 семестр лекционных занятий – 4, лабораторных занятий – 6. курсовое проектирование – 30 часов. Форма отчётности – экзамен.

Содержание учебного материала

Раздел 1 Основные понятия и технологии работы с реляционными базами данных

Тема 1 Модели данных

Понятие базы данных. Объекты, атрибуты, отношения. Физическое и логическое представления баз данных. Модели баз данных как способы отображения связей предметных областей. Основные модели баз данных. Уровни представления баз данных. Иерархическая, сетевая и реляционная модели данных.

Тема 2 Системы управления базами данных

Понятие систем управления базами данных (СУБД). Основные функции СУБД. Архитектура типичной СУБД. Главные компоненты СУБД. Классификация моделей СУБД. Модели ранних СУБД. Достоинства и недостатки ранних СУБД. Иерархические системы. Сетевые системы. Анализ современных СУБД как систем программирования. Понятие технологии «клиент-сервер». Методы обеспечения безопасности баз данных. Администрирование баз данных.

Тема 3 Реляционная модель данных

Понятие реляционной модели данных. Основные объекты реляционной модели. Целостность реляционных данных. Понятие первичных ключей таблиц реляционной СУБД. Понятие внешних ключей таблиц. Правила внешних ключей.

Раздел 2 Работа с современными промышленными СУБД

Тема 1 Методы поиска и анализа информации в базе данных

Операции запоминания и выборки данных. Сортировка и поиск данных. Оптимизация доступа к записям. Индексирование. Понятие индекса, виды индексов. Роль индексов при работе с одиночными и связанными таблицами.

Тема 2 СУБД MS Access. Работа в интерактивном режиме

Типы полей. Дополнительные параметры и ограничения, устанавливаемые для полей. Типы связей и их настройка. Правила совместимости типов при установке связей. Создание таблиц и схемы базы данных при помощи мастеров, шаблонов и конструкторов.

Тема 3 Средства диалогового построения запросов

Средства диалогового построения запросов в системах разработки приложений (бланки QBE). Перекрестные запросы. Группировка данных в запросах. Группировочные функции. Многотабличные запросы. Схема данных. Запросы действий.

Раздел 3 Функциональные зависимости и поиск данных

Тема 1 Проблема аномалии и задача нормализации данных

Аномалии включения и удаления данных и необходимость нормализации. Понятие о нормальной форме. 1, 2, 3 и 4 нормальные формы. Нормальная форма Бойса-Кодда. Приведение базы данных к нормализованному виду.

Тема 2 Использование языка запросов SQL для выборки данных

Использование языка SQL для выборки данных из таблиц. Общая структура оператора SELECT языка SQL. Указание состава полей и таблиц в операторе SELECT. Определение дополнительных имен (алиасов) таблиц и полей. Использование выражений в списке полей запроса.

Тема 3 Выборка данных из множества таблиц

Использование множества таблиц в запросах языка SQL на выборку данных. Реализация декартового произведения таблиц с помощью предложения WHERE. Самообъединение таблиц. Объединение таблиц с помощью операции JOIN. Левое внешнее объединение. Правое внешнее объединение. Вложенные операции JOIN.

Тема 4 Запросы обновления таблиц в языке запросов SQL

Запросы обновления таблиц. Создание одиночных записей с помощью оператора INSERT INTO. Использование подзапроса в операторе INSERT INTO для формирования записей, добавляемых в таблицу. Изменение значений полей с помощью оператора UPDATE. Использование подзапроса для определения группы изменяемых записей. Удаление записей из таблиц с помощью оператора DELETE. Использование подзапроса для определения группы удаляемых записей.

Раздел 4 Проектирование баз данных

Тема 1 Элементы проектирования баз данных

Этапы проектирования баз данных. Инфологическое проектирование. Функциональный подход к проектированию баз данных. Предметный подход к проектированию баз данных. Проектирование с использованием метода "сущность-связь". Определение требований к операционной обстановке. Выбор СУБД и других инструментальных программных средств. Логическое проектирование баз данных. Физическое проектирование баз данных. Автоматизация проектирования баз данных.

Тема 2 Проектирование баз данных

Определение состава таблиц. Виды взаимосвязей между таблицами реляционной базы данных: «один-к-одному», «один-к-многим», «многие-к-многим». Понятие родительской и дочерней таблицы. Удаление связей

«многие-к-многим» путем создания дополнительных отношений. Определение состава атрибутов отношений (таблиц). Нормализация отношений. Первая, вторая и третья нормальные формы. Достоинства и недостатки нормализации.

Тема 3 Описание структуры базы данных

Системы быстрой разработки приложений. Описание структур баз данных в существующих программных системах быстрой разработки приложений. Понятие NULL-значений в таблицах базы данных. Приемы использования NULL-значений в таблицах базы данных. Описание вычисляемых полей.

Раздел 5 Работа с базами данных из современных языков программирования

Тема 1 Язык описания данных

Понятия языка манипулирования данным и языка описания данных. Создание таблиц с помощью языка манипулирования данными (SELECT INTO).

Создание таблиц с помощью языка манипулирования данными (CREATE TABLE). Определение состава и типов полей, значений по умолчанию для столбцов таблицы, допустимости NULL-значений. Определение первичного ключа таблицы. Определение ограничений на множество допустимых значений данных, ограничений уникальности значений столбцов.

Тема 2 Ссылочная целостность данных

Понятие ссылочной целостности данных. Ограничения целостности данных. Поддержка ограничений целостности данных. Методы обеспечения целостности данных. Определение внешних ключей таблиц. Определение родительских ключей таблиц. Задание способов каскадных обновлений таблиц.

Тема 3 Оптимизация работы с базами данных

Модификация структуры таблицы. Уничтожение таблицы.

Назначение и использование индексов для таблиц базы данных. Уникальные и неуникальные индексы. Создание индексов по одному полю. Создание индексов по множеству полей. Оптимизация состава индексов. Уничтожение индексов.

Тема 4 Представления

Назначение представлений (просмотров). Создание представлений. Обновляемые и необновляемые представления. Изменение значений с помощью представлений. Контроль добавления записей в соответствии с условиями запроса представления. Контроль изменения записей в соответствии с условиями запроса представления. Уничтожение представлений.

Раздел 6 Работа с сетевыми базами данных

Тема 1 Особенности языка SQL для различных SQL-серверов

Дополнительные параметры оператора Select. Возможность использования подзапросов в качестве источника данных. Оператор Case. Дополнительные возможности в операторе добавления данных Insert. Дополнительные возможности в операторе обновления данных Update. Дополнительные возможности в операторе удаления данных Delete.

Тема 2 Создание таблиц баз данных в среде SQL-сервера

Способы создания таблиц баз данных в среде SQL-сервера. Понятие о типах данных. Выбор типов данных. Создание временных таблиц. Определение связей между таблицами. Определение внешних и родительских ключей таблиц. Определение рационального состава индексов таблиц. Создание индексов таблиц. Принципы создания индексов. Правила создания хорошего индекса.

Тема 3 Установка связи с таблицами баз данных

Использование средств ODBC. Описание ODBC-источников данных. Создание источников данных ODBC. Преимущества и недостатки использования ODBC. Установка связи с данными на SQL-сервере. Определение прав доступа к данным.

Тема 4 Создание и использование представлений в среде SQL-сервера

Создание представлений в среде SQL-сервера. Типы представлений в среде SQL-сервера. Испальзование представлений в среде SQL-сервера. Изменение значений с помощью представлений в среде SQL-сервера. Контроль добавления и изменения записей в соответствии с условиями запроса представления в среде SQL-сервера. Уничтожение представлений в среде SQL-сервера.

Тема 5 Хранимые процедуры

Назначение хранимых процедур. Язык написания хранимых процедур. Определение хранимых процедур. Определение и использование параметров хранимых процедур. Вызов хранимых процедур из приложения. Отображение и редактирование хранимых процедур. Изменение и удаление хранимых процедур. Компиляция процедуры и преобразование имен.

Тема 6 Создание и использование курсоров

Понятие курсора. Клиентские и серверные курсоры. Использование курсоров SQL-сервера: объявление курсора, открытие курсора, выборка данных из курсора, закрытие курсора, аннулирование курсора. Примеры использования курсоров. Использование вложенных курсоров. Работа с курсорами в клиентских приложениях для организации циклов обработки результатов запросов.

Тема 7 Триггеры SQL-сервера

Понятие о триггерах SQL-сервера. Назначение триггеров: поддержка ссылочной целостности базы данных, ведение журналов изменений базы данных, реализация функций контроля за корректностью данных, накопление итоговой информации. Виды триггеров по типам операций с базой данных. Виды триггеров по обработке отдельных строк или запросов. Ограничения, налагаемые триггерами.

Тема 8 Использование триггеров SQL-сервера

Создание триггеров. Удаление триггеров. Отображение информации о триггере. Использование триггеров для операторов Insert и Update. Использование триггеров для оператора Delete. Использование вложенных триггеров.

Тема 9 Транзакции

Понятие транзакций. Понятие блокировки. Откат транзакций в триггерах. Специальные методы управления транзакциями. Реализация триггеров в процессе создания клиент-серверного приложения баз данных.

Тема 10 Основные понятия MySQL

Структура языка SQL принятого в MySQL. Основные понятия MySQL. Примеры синтаксиса. Типы столбцов в таблицах. Манипуляции с данными в таблицах. Проблемы защиты и система привилегий пакета.

Тема 11 Администрирование сетевых БД

Функции администратора БД. Работа с пользователями, установление стадартов и процедур. Задачи администратора БД. Целостность БД. Обработка транзакций, контроль параллельной обработки.

Тема 12 Защита сетевых БД

Идентификация пользователя, проверка полномочий и представления данных, шифровка. Восстановление БД, источники отказов, процедуры восстановления, протоколирование с отложенными обновлениями, протоколирование с немедленными обновлениями, контрольные точки.

Тема 13 Совместное использование данных и базы данных

Совместное использование данных пользователями разных уровней. Совместное использование данных региональными отделениями. Критерии эффективности БД. БД и контроль управления.

Раздел 7 Базы данных в Интернет

Тема 1 Сервер Web как ядро приложений для Интернет

Основы работы сервера Web. Пассивные и активные серверы Web. Активность на стороне клиента. Интеграция серверов Web и SQL Server. Проект Интернет магазина. Установка программ и подготовка к работе.

Тема 2 Связь приложений с базами данных через ODBC

Программный интерфейс ODBC.Обработка ошибок. Программа ODBCAPP. Запуск хранимых процедур.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА (дневная форма обучения)

Номер раздела, темы, занятия	Название раздела, темы, занятия; перечень изучаемых вопросов	Количество аудиторных часов					Кол-во часов УСР	Формы контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Иное		
1	2	3	4	5	6	7	8	9
1.	**ОСНОВНЫЕ ПОНЯТИЯ И ТЕХНОЛОГИИ РАБОТЫ С РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ (10 Ч.)**	**6**			**4**			
1.1.	МОДЕЛИ ДАННЫХ 1 Понятие базы данных. 2 Объекты, атрибуты, отношения. Физическое и логическое представления баз данных. 3 Модели баз данных как способы отображения связей предметных областей. 4 Основные модели баз данных. 5 Уровни представления баз данных. Иерархическая, сетевая и реляционная модели данных	**2**						
1.2	ПОНЯТИЕ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫ) 1 Понятие систем управления базами данных (СУБД). 2 Основные функции СУБД. 3 Архитектура типичной СУБД. 4 Главные компоненты СУБД. Классификация моделей СУБД.	**2**			**4**			Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
1.3	Реляционная модель данных 1 Понятие реляционной модели данных. 2 Основные объекты реляционной модели. 3 Целостность реляционных данных. 4 Понятие первичных ключей таблиц реляционной СУБД. 5 Понятие внешних ключей таблиц. 6 Правила внешних ключей	**2**						
2.	**РАБОТА С СОВРЕМЕННЫМИ ПРОМЫШЛЕННЫМИ СУБД (8 Ч.)**	**4**			**2**		**2**	
2.1.	Методы поиска и анализа информации в базе данных **1 Операции запоминания и выборки данных.** **2 Сортировка и поиск данных.** **3 Оптимизация доступа к записям.** **4 Индексирование.** **5 Понятие индекса, виды индексов.** **6 Роль индексов при работе с одиночными и связанными таблицами.** .	**2**						
2.2.	СУБД MS Access. Работа в интерактивном режиме **1 Типы полей.** **2 Дополнительные параметры и ограничения, устанавливаемые для полей.** **3 Типы связей и их настройка.** **4 Правила совместимости типов при установке связей.** **5 Создание таблиц и схемы базы данных при помощи мастеров, шаблонов и конструкторов.**						**2**	
2.3.	Средства диалогового построения запросов **1 Средства диалогового построения запросов в системах разработки приложений (бланки QBE).** **2 Перекрестные запросы.** **3 Группировка данных в запросах.** **4 Группировочные функции.**	**2**			**2**			Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
	5 Многотабличные запросы. **6 Схема данных.** **7 Запросы действий.** .							
3.	**ФУНКЦИОНАЛЬНЫЕ ЗАВИСИМОСТИ И ПОИСК ДАННЫХ (12 ч.)**	**6**			**4**		**2**	
3.1.	Проблема аномалии и задача нормализации данных **1 Аномалии включения и удаления данных и необходимость нормализации.** **2 Понятие о нормальной форме. 1, 2, 3 и 4 нормальные формы.** **3 Нормальная форма Бойса-Кодда.** 4 **Приведение базы данных к нормализованному виду.**						**2**	
3.2.	Использование языка запросов SQL для выборки данных 1 Использование языка SQL для выборки данных из таблиц. 2 Общая структура оператора SELECT языка SQL. 3 Указание состава полей и таблиц в операторе SELECT. 4 Определение дополнительных имен (алиасов) таблиц и полей. 5 Использование выражений в списке полей запроса.	**2**			**2**			Защита отчета по лаб. работе
3.3.	Выборка данных из множества таблиц 1 Использование множества таблиц в запросах языка SQL на выборку данных. 2 Реализация декартового произведения таблиц с помощью предложения WHERE. 3 Самообъединение таблиц. 4 Объединение таблиц с помощью операции JOIN. 5 Левое внешнее объединение. 6 Правое внешнее объединение.	**2**			**2**			Защита отчета по лаб. работе
3.4.	Запросы обновления таблиц в языке запросов SQL **1 Запросы обновления таблиц.** **2 Создание одиночных записей с помощью оператора INSERT INTO.** **3 Использование подзапроса в операторе INSERT INTO для формирования записей, добавляемых в таблицу.** **4 Изменение значений полей с помощью оператора UPDATE.** **5 Использование подзапроса для определения группы изменяемых записей.** **6 Удаление записей из таблиц с помощью оператора DELETE.** **7 Использование подзапроса для определения группы удаляемых записей.**	**2**						

1	2	3	4	5	6	7	8	9
4.	**ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ (10 Ч.)**	**4**			**4**		**2**	
4.1.	Элементы проектирования баз данных 1 Определение требований к операционной обстановке. 2 Выбор СУБД и других инструментальных программных средств. 3 Логическое проектирование баз данных. 4 Физическое проектирование баз данных. 5 Автоматизация проектирования баз данных. .	**2**						
4.2.	Проектирование баз данных 1. Определение состава таблиц. 2. Виды взаимосвязей между таблицами реляционной базы данных: «один-к-одному», «один-к-многим», «многие-к-многим». 3. Понятие родительской и дочерней таблицы. 4. Удаление связей «многие-к-многим» путем создания дополнительных отношений. 5. Определение состава атрибутов отношений (таблиц). 6. Нормализация отношений. 7. Первая, вторая и третья нормальные формы. 8. Достоинства и недостатки нормализации.	**2**			**4**			Защита отчета по лаб. работе
4.3.	Описание структуры базы данных 1. Системы быстрой разработки приложений. 2. Описание структур баз данных в существующих программных системах быстрой разработки приложений. 3. Понятие NULL-значений в таблицах базы данных. 4. Приемы использования NULL-значений в таблицах базы данных. 5. Описание вычисляемых полей. .						**2**	
5	**РАБОТА С БАЗАМИ ДАННЫХ ИЗ СОВРЕМЕННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ (12Ч)**	**6**			**4**		**2**	

1	2	3	4	5	6	7	8	9
5.1	Язык описания данных 1 Понятия языка манипулирования данным и языка описания данных. 2 Создание таблиц с помощью языка манипулирования данными (SELECT INTO). 3 Создание таблиц с помощью языка манипулирования данными (CREATE TABLE). 4 Определение состава и типов полей, значений по умолчанию для столбцов таблицы, допустимости NULL-значений. 5 Определение первичного ключа таблицы. **6** Определение ограничений на множество допустимых значений данных, ограничений уникальности значений столбцов	**2**						
5.2	Ссылочная целостность данных 1.Понятие ссылочной целостности данных. 2.Ограничения целостности данных. 3.Поддержка ограничений целостности данных. 4.Методы обеспечения целостности данных. 5.Определение внешних ключей таблиц. 6.Определение родительских ключей таблиц. 7.Задание способов каскадных обновлений таблиц.						**2**	
5.3	Оптимизация работы с базами данных 1 Модификация структуры таблицы. 2 Уничтожение таблицы. 3 Назначение и использование индексов для таблиц базы данных. 4 Уникальные и неуникальные индексы. 5 Создание индексов по одному полю. 6 Создание индексов по множеству полей. 7 Оптимизация состава индексов. 8 Уничтожение индексов.	**2**			**2**			Защита отчета по лаб. работе
5.4	Представления 1 Назначение представлений (просмотров). 2 Создание представлений. 3 Обновляемые и необновляемые представления. 4 Изменение значений с помощью представлений. 5 Контроль добавления записей в соответствии с условиями запроса	**2**			**2**			Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
	представления. 6 Контроль изменения записей в соответствии с условиями запроса представления. 7 Уничтожение представлений.							
	Всего по дисциплине 3й семестр	**26**			**18**		**8**	Зачет
6.	**РАБОТА С СЕТЕВЫМИ БАЗАМИ ДАННЫХ (76 Ч.)**	**16**			**50**		**10**	
6.1	Особенности языка SQL для различных SQL-серверов 1 Дополнительные параметры оператора Select. 2 Возможность использования подзапросов в качестве источника данных. 3 Оператор Case. 4 Дополнительные возможности в операторе добавления данных Insert. 5 Дополнительные возможности в операторе обновления данных Update. 6 Дополнительные возможности в операторе удаления данных Delete..	**2**			**4**			Защита отчета по лаб. работе
6.2	Создание таблиц баз данных в среде SQL-сервера 1 Способы создания таблиц баз данных в среде SQL-сервера. 2 Понятие о типах данных. 3 Выбор типов данных. 4 Создание временных таблиц. 5 Определение связей между таблицами. 6 Определение внешних и родительских ключей таблиц. 7 Определение рационального состава индексов таблиц. 8 Создание индексов таблиц. 9 Принципы создания индексов. 10 Правила создания хорошего индекса.				**4**		**2**	Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
6.3	Установка связи с таблицами баз данных 1 Использование средств ODBC. 2 Описание ODBC-источников данных. 3 Создание источников данных ODBC. 4 Преимущества и недостатки использования ODBC. 5 Установка связи с данными на SQL-сервере. 6 Определение прав доступа к данным.	**2**			**4**			Защита отчета по лаб. работе
6.4	Создание и использование представлений в среде SQL-сервера 1 Создание представлений в среде SQL-сервера. 2 Типы представлений в среде SQL-сервера. 3 Использование представлений в среде SQL-сервера. 4 Изменение значений с помощью представлений в среде SQL-сервера. 5 Контроль добавления и изменения записей в соответствии с условиями запроса представления в среде SQL-сервера. 6 Уничтожение представлений в среде SQL-сервера. .	**2**			**4**			Защита отчета по лаб. работе
6.5	Хранимые процедуры 1 Назначение хранимых процедур. 2 Язык написания хранимых процедур. 3 Определение хранимых процедур. 4 Определение и использование параметров хранимых процедур. Вызов хранимых процедур из приложения. 5 Отображение и редактирование хранимых процедур. Изменение и удаление хранимых процедур. **6 Компиляция процедуры и преобразование имен.**	**2**			**4**			Защита отчета по лаб. работе
6.6	Создание и использование курсоров 1 Понятие курсора. 2 Клиентские и серверные курсоры. 3 Использование курсоров SQL-сервера: объявление курсора, открытие курсора, выборка данных из курсора, закрытие курсора, аннулирование курсора. 4 Примеры использования курсоров. 5 Использование вложенных курсоров. 6 Работа с курсорами в клиентских приложениях для организации циклов обработки результатов запросов.				**4**		**2**	Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
6.7	Триггеры SQL-сервера 1 Понятие о триггерах SQL-сервера. 2 Назначение триггеров: поддержка ссылочной целостности базы данных, ведение журналов изменений базы данных, реализация функций контроля за корректностью данных, накопление итоговой информации. 3 Виды триггеров по типам операций с базой данных. 4 Виды триггеров по обработке отдельных строк или запросов. Ограничения, налагаемые триггерами	**2**			**4**			Защита отчета по лаб. работе
6.8	Использование триггеров SQL-сервера 1 Создание триггеров. 2 Удаление триггеров. 3 Отображение информации о триггере. 4 Использование триггеров для операторов Insert и Update. 5 Использование триггеров для оператора Delete. 6 Использование вложенных триггеров.	**2**			**4**			Защита отчета по лаб. работе
6.9	Транзакции 1 Понятие транзакций. 2 Понятие блокировки. 3 Откат транзакций в триггерах. 4 Специальные методы управления транзакциями. 5 Реализация триггеров в процессе создания клиент-серверного приложения баз данных.	**2**			**4**			Защита отчета по лаб. работе
6.10	Основные понятия MySQL **1. Структура языка SQL принятого в MySQL.** **2. Основные понятия MySQL.** **3. Примеры синтаксиса.** **4. Типы столбцов в таблицах.** **5. Манипуляции с данными в таблицах** **6. Проблемы защиты и система привилегий пакета.**				**4**		**2**	Защита отчета по лаб. работе
6.11	**Администрирование сетевых БД** 1 Функции администратора БД. 2 Работа с пользователями, установление стадартов и процедур. 3 Задачи администратора БД. 4 Целостность БД. **5 Обработка транзакций, контроль параллельной обработки**.	**2**			**4**			Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
6.12	**Защита сетевых БД** 1 Идентификация пользователя. 2 Проверка полномочий и представления данных, шифровка. 3 Восстановление БД, источники отказов, процедуры восстановления, протоколирование с отложенными обновлениями, протоколирование с немедленными обновлениями, контрольные точки.				**4**		**2**	Защита отчета по лаб. работе
6.13	**Совместное использование данных и базы данных** 1 Совместное использование данных пользователями разных уровней. 2 Совместное использование данных региональными отделениями. 3 Критерии эффективности БД. 4 БД и контроль управления.				**2**		**2**	Защита отчета по лаб. работе
7	**БАЗЫ ДАННЫХ В ИНТЕРНЕТ (12 Ч.)**	**4**			**8**			
7.1	**Сервер Web как ядро приложений для Интернет** 1 Основы работы сервера Web. 2 Пассивные и активные серверы Web. 3 Активность на стороне клиента. 4 Интеграция серверов Web и SQL Server. **5** Проект Интернет магазина. Установка программ и подготовка к работе.	**2**			**4**			Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
7.2	**Связь приложений с базами данных через ODBC** 1. Программный интерфейс ODBC. 2. Обработка ошибок. 3. Программа ODBCAPP. 4. Запуск хранимых процедур.	**2**			**4**			Защита отчета по лаб. работе
	Всего по дисциплине 5й семестр	**20**			**58**		**10**	экзамен
	Всего по дисциплине	**46**			**76**		**18**	

Старший преподаватель кафедры АСОИ В.Н.Леванцов

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА (заочная форма обучения)

Номер раздела, темы, занятия	Название раздела, темы, занятия; перечень изучаемых вопросов	Количество аудиторных часов					Кол-во часов УСР	Формы контроля знаний
		Лекции	Практические занятия	Семинарские занятия	занятия	Иное		
1	2	3	4	5	6	7	8	9
1.	**ОСНОВНЫЕ ПОНЯТИЯ И ТЕХНОЛОГИИ РАБОТЫ С РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ (2 Ч.)**	**2**						
1.1.	МОДЕЛИ ДАННЫХ 1 Понятие базы данных. 2 Объекты, атрибуты, отношения. Физическое и логическое представления баз данных. 3 Модели баз данных как способы отображения связей предметных областей. 4 Основные модели баз данных. 5 Уровни представления баз данных. Иерархическая, сетевая и реляционная модели данных	Самостоятельное изучение						
1.2	ПОНЯТИЕ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫ) 1 Понятие систем управления базами данных (СУБД). 2 Основные функции СУБД. 3 Архитектура типичной СУБД. 4 Главные компоненты СУБД. Классификация моделей СУБД.	**2**						

1	2	3	4	5	6	7	8	9
1.3	Реляционная модель данных 1 Понятие реляционной модели данных. 2 Основные объекты реляционной модели. 3 Целостность реляционных данных. 4 Понятие первичных ключей таблиц реляционной СУБД. 5 Понятие внешних ключей таблиц. 6 Правила внешних ключей	Самостоятельное изучение						
2.	**РАБОТА С СОВРЕМЕННЫМИ ПРОМЫШЛЕННЫМИ СУБД (4 Ч.)**	**2**			**2**			
2.1.	Методы поиска и анализа информации в базе данных **1 Операции запоминания и выборки данных.** **2 Сортировка и поиск данных.** **3 Оптимизация доступа к записям.** **4 Индексирование.** **5 Понятие индекса, виды индексов.** **6 Роль индексов при работе с одиночными и связанными таблицами.** .	**2**						
2.2.	СУБД MS Access. Работа в интерактивном режиме **6 Типы полей.** **1 Дополнительные параметры и ограничения, устанавливаемые для полей.** **2 Типы связей и их настройка.** **3 Правила совместимости типов при установке связей.** **4 Создание таблиц и схемы базы данных при помощи мастеров, шаблонов и конструкторов.**	Самостоятельное изучение						
2.3.	Средства диалогового построения запросов **1 Средства диалогового построения запросов в системах разработки приложений (бланки QBE).** **2 Перекрестные запросы.** **3 Группировка данных в запросах.** **4 Группировочные функции.**	Самостоятельное изучение			**2**			

1	2	3	4	5	6	7	8	9
	5 Многотабличные запросы. **6 Схема данных.** **7 Запросы действий.** .							
3.	ФУНКЦИОНАЛЬНЫЕ ЗАВИСИМОСТИ И ПОИСК ДАННЫХ **(4ч.)**	**2**			**2**			
3.1.	Проблема аномалии и задача нормализации данных **1 Аномалии включения и удаления данных и необходимость нормализации.** **2 Понятие о нормальной форме. 1, 2, 3 и 4 нормальные формы.** **3 Нормальная форма Бойса-Кодда.** 4 **Приведение базы данных к нормализованному виду.**	Самостоятельное изучение						
3.2.	Использование языка запросов SQL для выборки данных 6 Использование языка SQL для выборки данных из таблиц. 1 Общая структура оператора SELECT языка SQL. 2 Указание состава полей и таблиц в операторе SELECT. 3 Определение дополнительных имен (алиасов) таблиц и полей. 4 Использование выражений в списке полей запроса.	**2**			**2**			Защита отчета по лаб. работе
3.3.	Выборка данных из множества таблиц 1 Использование множества таблиц в запросах языка SQL на выборку данных. 2 Реализация декартового произведения таблиц с помощью предложения WHERE. 3 Самообъединение таблиц. 4 Объединение таблиц с помощью операции JOIN. 5 Левое внешнее объединение. 6 Правое внешнее объединение.	Самостоятельное изучение						
3.4.	Запросы обновления таблиц в языке запросов SQL **8 Запросы обновления таблиц.** **1 Создание одиночных записей с помощью оператора INSERT INTO.** **2 Использование подзапроса в операторе INSERT INTO для формирования записей, добавляемых в таблицу.** **3 Изменение значений полей с помощью оператора UPDATE.** **4 Использование подзапроса для определения группы изменяемых записей.** **5 Удаление записей из таблиц с помощью оператора DELETE.** **6 Использование подзапроса для определения группы удаляемых записей.**	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
4.	**ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ (4 Ч.)**	**2**			**2**			
4.1.	Элементы проектирования баз данных 1 Определение требований к операционной обстановке. 2 Выбор СУБД и других инструментальных программных средств. 3 Логическое проектирование баз данных. 4 Физическое проектирование баз данных. 5 Автоматизация проектирования баз данных. .	Самостоятельное изучение						
4.2.	Проектирование баз данных 1. Определение состава таблиц. 2. Виды взаимосвязей между таблицами реляционной базы данных: «один-к-одному», «один-к-многим», «многие-к-многим». 3. Понятие родительской и дочерней таблицы. 4. Удаление связей «многие-к-многим» путем создания дополнительных отношений. 5. Определение состава атрибутов отношений (таблиц). 6. Нормализация отношений. 7. Первая, вторая и третья нормальные формы. 8. Достоинства и недостатки нормализации.	**2**			**2**			Защита отчета по лаб. работе
4.3.	Описание структуры базы данных 1. Системы быстрой разработки приложений. 2. Описание структур баз данных в существующих программных системах быстрой разработки приложений. 3. Понятие NULL-значений в таблицах базы данных. 4. Приемы использования NULL-значений в таблицах базы данных. 5. Описание вычисляемых полей. .	Самостоятельное изучение						
5	**РАБОТА С БАЗАМИ ДАННЫХ ИЗ СОВРЕМЕННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ (4Ч)**	**2**			**2**			

1	2	3	4	5	6	7	8	9
5.1	Язык описания данных 7 Понятия языка манипулирования данным и языка описания данных. 1 Создание таблиц с помощью языка манипулирования данными (SELECT INTO). 2 Создание таблиц с помощью языка манипулирования данными (CREATE TABLE). 3 Определение состава и типов полей, значений по умолчанию для столбцов таблицы, допустимости NULL-значений. 4 Определение первичного ключа таблицы. **5** Определение ограничений на множество допустимых значений данных, ограничений уникальности значений столбцов	Самостоятельное изучение						
5.2	Ссылочная целостность данных 1. Понятие ссылочной целостности данных. 2. Ограничения целостности данных. 3. Поддержка ограничений целостности данных. 4. Методы обеспечения целостности данных. 5. Определение внешних ключей таблиц. 6. Определение родительских ключей таблиц. 7. Задание способов каскадных обновлений таблиц.	Самостоятельное изучение						
5.3	Оптимизация работы с базами данных 1 Модификация структуры таблицы. 2 Уничтожение таблицы. 3 Назначение и использование индексов для таблиц базы данных. 4 Уникальные и неуникальные индексы. 5 Создание индексов по одному полю. 6 Создание индексов по множеству полей. 7 Оптимизация состава индексов. 8 Уничтожение индексов.	Самостоятельное изучение						
5.4	Представления 1 Назначение представлений (просмотров). 2 Создание представлений. 3 Обновляемые и необновляемые представления. 4 Изменение значений с помощью представлений. 5 Контроль добавления записей в соответствии с условиями запроса	**2**			**2**			Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
	представления. 6 Контроль изменения записей в соответствии с условиями запроса представления. 7 Уничтожение представлений.							
	Всего по дисциплине 3й семестр	**10**			**8**			Зачет
6.	**РАБОТА С СЕТЕВЫМИ БАЗАМИ ДАННЫХ (76 Ч.)**	**8**			**6**			
6.1	Особенности языка SQL для различных SQL-серверов 7 Дополнительные параметры оператора Select. 1 Возможность использования подзапросов в качестве источника данных. 2 Оператор Case. 3 Дополнительные возможности в операторе добавления данных Insert. 4 Дополнительные возможности в операторе обновления данных Update. 5 Дополнительные возможности в операторе удаления данных Delete..	**2**						
6.2	Создание таблиц баз данных в среде SQL-сервера 1 Способы создания таблиц баз данных в среде SQL-сервера. 2 Понятие о типах данных. 3 Выбор типов данных. 4 Создание временных таблиц. 5 Определение связей между таблицами. 6 Определение внешних и родительских ключей таблиц. 7 Определение рационального состава индексов таблиц. 8 Создание индексов таблиц. 9 Принципы создания индексов. 10 Правила создания хорошего индекса.	Самостоятельное изучение			**2**			Защита отчета по лаб. работе

1	2	3	4	5	6	7	8	9
6.3	Установка связи с таблицами баз данных 1 Использование средств ODBC. 2 Описание ODBC-источников данных. 3 Создание источников данных ODBC. 4 Преимущества и недостатки использования ODBC. 5 Установка связи с данными на SQL-сервере. 6 Определение прав доступа к данным.	**2**						
6.4	Создание и использование представлений в среде SQL-сервера 1 Создание представлений в среде SQL-сервера. 2 Типы представлений в среде SQL-сервера. 3 Использование представлений в среде SQL-сервера. 4 Изменение значений с помощью представлений в среде SQL-сервера. 5 Контроль добавления и изменения записей в соответствии с условиями запроса представления в среде SQL-сервера. 6 Уничтожение представлений в среде SQL-сервера. .	Самостоятельное изучение						
6.5	Хранимые процедуры 1 Назначение хранимых процедур. 2 Язык написания хранимых процедур. 3 Определение хранимых процедур. 4 Определение и использование параметров хранимых процедур. Вызов хранимых процедур из приложения. 5 Отображение и редактирование хранимых процедур. Изменение и удаление хранимых процедур. **6 Компиляция процедуры и преобразование имен.**	**2**			**2**			Защита отчета по лаб. работе
6.6	Создание и использование курсоров 1 Понятие курсора. 2 Клиентские и серверные курсоры. 3 Использование курсоров SQL-сервера: объявление курсора, открытие курсора, выборка данных из курсора, закрытие курсора, аннулирование курсора. 4 Примеры использования курсоров. 5 Использование вложенных курсоров. 6 Работа с курсорами в клиентских приложениях для организации циклов обработки результатов запросов.	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
6.7	Триггеры SQL-сервера 1 Понятие о триггерах SQL-сервера. 2 Назначение триггеров: поддержка ссылочной целостности базы данных, ведение журналов изменений базы данных, реализация функций контроля за корректностью данных, накопление итоговой информации. 3 Виды триггеров по типам операций с базой данных. 4 Виды триггеров по обработке отдельных строк или запросов. Ограничения, налагаемые триггерами	**2**			**2**			
6.8	Использование триггеров SQL-сервера 1 Создание триггеров. 2 Удаление триггеров. 3 Отображение информации о триггере. 4 Использование триггеров для операторов Insert и Update. 5 Использование триггеров для оператора Delete. 6 Использование вложенных триггеров.	Самостоятельное изучение						
6.9	Транзакции 1 Понятие транзакций. 2 Понятие блокировки. 3 Откат транзакций в триггерах. 4 Специальные методы управления транзакциями. 5 Реализация триггеров в процессе создания клиент-серверного приложения баз данных.	Самостоятельное изучение						
6.10	Основные понятия MySQL **1. Структура языка SQL принятого в MySQL.** **2. Основные понятия MySQL.** **3. Примеры синтаксиса.** **4. Типы столбцов в таблицах.** **5. Манипуляции с данными в таблицах** **6. Проблемы защиты и система привилегий пакета.**	Самостоятельное изучение						
6.11	**Администрирование сетевых БД** 1 Функции администратора БД. 2 Работа с пользователями, установление стадартов и процедур. 3 Задачи администратора БД. 4 Целостность БД. **5 Обработка транзакций, контроль параллельной обработки**.	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
6.12	**Защита сетевых БД** 1 Идентификация пользователя. 2 Проверка полномочий и представления данных, шифровка. 3 Восстановление БД, источники отказов, процедуры восстановления, протоколирование с отложенными обновлениями, протоколирование с немедленными обновлениями, контрольные точки.	Самостоятельное изучение						
6.13	**Совместное использование данных и базы данных** 1 Совместное использование данных пользователями разных уровней. 2 Совместное использование данных региональными отделениями. 3 Критерии эффективности БД. 4 БД и контроль управления.	Самостоятельное изучение						
7	**БАЗЫ ДАННЫХ В ИНТЕРНЕТ (12 Ч.)**	**2**			**2**			
7.1	**Сервер Web как ядро приложений для Интернет** 1 Основы работы сервера Web. 2 Пассивные и активные серверы Web. 3 Активность на стороне клиента. 4 Интеграция серверов Web и SQL Server. **5** Проект Интернет магазина. Установка программ и подготовка к работе.	**2**						

1	2	3	4	5	6	7	8	9
7.2	**Связь приложений с базами данных через ODBC** 1. Программный интерфейс ODBC. 2. Обработка ошибок. 3. Программа ODBCAPP. 4. Запуск хранимых процедур.	Самостоятельное изучение			**2**			Защита отчета по лаб. работе
	Всего по дисциплине 5й семестр	**2**			**8**			экзамен
	Всего по дисциплине	**20**			**16**			

Старший преподаватель кафедры АСОИ В.Н.Леванцов

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА (заочная дистанционная интегрированная форма обучения на основе среднего специального образования)

Номер раздела, темы, занятия	Название раздела, темы, занятия; перечень изучаемых вопросов	Количество аудиторных часов					Кол-во часов УСР	Формы контроля знаний
		Лекции	Практические занятия	Семинарские занятия	занятия	Иное		
1	2	3	4	5	6	7	8	9
1.	**ОСНОВНЫЕ ПОНЯТИЯ И ТЕХНОЛОГИИ РАБОТЫ С РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ (2 Ч.)**	**2**						
1.1.	МОДЕЛИ ДАННЫХ 1 Понятие базы данных. 2 Объекты, атрибуты, отношения. Физическое и логическое представления баз данных. 3 Модели баз данных как способы отображения связей предметных областей. 4 Основные модели баз данных. 5 Уровни представления баз данных. Иерархическая, сетевая и реляционная модели данных	Самостоятельное изучение						
1.2	ПОНЯТИЕ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫ) 1 Понятие систем управления базами данных (СУБД). 2 Основные функции СУБД. 3 Архитектура типичной СУБД. 4 Главные компоненты СУБД.	**2**						

1	2	3	4	5	6	7	8	9
	Классификация моделей СУБД.							
1.3	Реляционная модель данных 1 Понятие реляционной модели данных. 2 Основные объекты реляционной модели. 3 Целостность реляционных данных. 4 Понятие первичных ключей таблиц реляционной СУБД. 5 Понятие внешних ключей таблиц. 6 Правила внешних ключей	Самостоятельное изучение						
	Всего по дисциплине 5 й семестр	**2**						
2.	**РАБОТА С СОВРЕМЕННЫМИ ПРОМЫШЛЕННЫМИ СУБД (2 Ч.)**				**2**			
2.1.	Методы поиска и анализа информации в базе данных **1 Операции запоминания и выборки данных.** **2 Сортировка и поиск данных.** **3 Оптимизация доступа к записям.** **4 Индексирование.** **5 Понятие индекса, виды индексов.** **6 Роль индексов при работе с одиночными и связанными таблицами.** .	Самостоятельное изучение						
2.2.	СУБД MS Access. Работа в интерактивном режиме **7 Типы полей.** **1 Дополнительные параметры и ограничения, устанавливаемые для полей.** **2 Типы связей и их настройка.** **3 Правила совместимости типов при установке связей.** **4 Создание таблиц и схемы базы данных при помощи мастеров, шаблонов и**	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
	конструкторов.							
2.3.	Средства диалогового построения запросов **1 Средства диалогового построения запросов в системах разработки приложений (бланки QBE).** **2 Перекрестные запросы.** **3 Группировка данных в запросах.** **4 Группировочные функции.** **5 Многотабличные запросы.** **6 Схема данных.** **7 Запросы действий.** .	Самостоятельное изучение						
3.	ФУНКЦИОНАЛЬНЫЕ ЗАВИСИМОСТИ И ПОИСК ДАННЫХ **(4ч.)**	**2**			**2**			
3.1.	Проблема аномалии и задача нормализации данных **1 Аномалии включения и удаления данных и необходимость нормализации.** **2 Понятие о нормальной форме. 1, 2, 3 и 4 нормальные формы.** **3 Нормальная форма Бойса-Кодда.** 4 **Приведение базы данных к нормализованному виду.**	Самостоятельное изучение						
3.2.	Использование языка запросов SQL для выборки данных 1 Использование языка SQL для выборки данных из таблиц 2 Общая структура оператора SELECT языка SQL. 3 Указание состава полей и таблиц в операторе SELECT. 4 Определение дополнительных имен (алиасов) таблиц и полей. 5 Использование выражений в списке полей запроса.	**2**			**2**			Защита отчета по лаб. работе
3.3.	Выборка данных из множества таблиц 1 Использование множества таблиц в запросах языка SQL на выборку данных. 2 Реализация декартового произведения таблиц с помощью предложения WHERE. 3 Самообъединение таблиц. 4 Объединение таблиц с помощью операции JOIN. 5 Левое внешнее объединение. 6 Правое внешнее объединение.	Самостоятельное изучение						
3.4.	Запросы обновления таблиц в языке запросов SQL **1 Запросы обновления таблиц.**	Самостоятельное						

1	2	3	4	5	6	7	8	9
	2 Создание одиночных записей с помощью оператора INSERT INTO. **3 Использование подзапроса в операторе INSERT INTO для формирования записей, добавляемых в таблицу.** **4 Изменение значений полей с помощью оператора UPDATE.** **5 Использование подзапроса для определения группы изменяемых записей.** **6 Удаление записей из таблиц с помощью оператора DELETE.** **7 Использование подзапроса для определения группы удаляемых записей.**	изучение						
4.	**ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ (4 Ч.)**	**2**			**2**			
4.1.	Элементы проектирования баз данных 1 Определение требований к операционной обстановке. 2 Выбор СУБД и других инструментальных программных средств. 3 Логическое проектирование баз данных. 4 Физическое проектирование баз данных. 5 Автоматизация проектирования баз данных. .	Самостоятельное изучение						
4.2.	Проектирование баз данных 1. Определение состава таблиц. 2. Виды взаимосвязей между таблицами реляционной базы данных: «один-к-одному», «один-к-многим», «многие-к-многим». 3. Понятие родительской и дочерней таблицы. 4. Удаление связей «многие-к-многим» путем создания дополнительных отношений. 5. Определение состава атрибутов отношений (таблиц). 6. Нормализация отношений. 7. Первая, вторая и третья нормальные формы. 8. Достоинства и недостатки нормализации.	**2**			**2**			Защита отчета по лаб. работе
4.3.	Описание структуры базы данных 1. Системы быстрой разработки приложений. 2. Описание структур баз данных в существующих программных системах быстрой разработки приложений. 3. Понятие NULL-значений в таблицах базы данных. 4. Приемы использования NULL-значений в таблицах базы данных.	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
	5. Описание вычисляемых полей. .							
5	**РАБОТА С БАЗАМИ ДАННЫХ ИЗ СОВРЕМЕННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ)**							
5.1	Язык описания данных 1 Понятия языка манипулирования данным и языка описания данных. 2 Создание таблиц с помощью языка манипулирования данными (SELECT INTO). 3 Создание таблиц с помощью языка манипулирования данными (CREATE TABLE). 4 Определение состава и типов полей, значений по умолчанию для столбцов таблицы, допустимости NULL-значений. 5 Определение первичного ключа таблицы. **6** Определение ограничений на множество допустимых значений данных, ограничений уникальности значений столбцов	Самостоятельное изучение						
5.2	Ссылочная целостность данных 1.Понятие ссылочной целостности данных. 2.Ограничения целостности данных. 3.Поддержка ограничений целостности данных. 4.Методы обеспечения целостности данных. 5.Определение внешних ключей таблиц. 6.Определение родительских ключей таблиц. 7.Задание способов каскадных обновлений таблиц.	Самостоятельное изучение						
5.3	Оптимизация работы с базами данных 1 Модификация структуры таблицы. 2 Уничтожение таблицы. 3 Назначение и использование индексов для таблиц базы данных. 4 Уникальные и неуникальные индексы. 5 Создание индексов по одному полю. 6 Создание индексов по множеству полей.	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
	7 Оптимизация состава индексов. 8 Уничтожение индексов.							
5.4	Представления 1 Назначение представлений (просмотров). 2 Создание представлений. 3 Обновляемые и необновляемые представления. 4 Изменение значений с помощью представлений. 5 Контроль добавления записей в соответствии с условиями запроса представления. 6 Контроль изменения записей в соответствии с условиями запроса представления. 7 Уничтожение представлений.	Самостоятельное изучение						
	Всего по дисциплине 6 й семестр	**4**			**6**			Зачет
6.	**РАБОТА С СЕТЕВЫМИ БАЗАМИ ДАННЫХ (8Ч.)**	**4**			**4**			
6.1	Особенности языка SQL для различных SQL-серверов 1 Дополнительные параметры оператора Select. 2 Возможность использования подзапросов в качестве источника данных. 3 Оператор Case. 4 Дополнительные возможности в операторе добавления данных Insert 5 Дополнительные возможности в операторе обновления данных Update. 6 Дополнительные возможности в операторе удаления данных Delete..	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
6.2	Создание таблиц баз данных в среде SQL-сервера 1 Способы создания таблиц баз данных в среде SQL-сервера. 2 Понятие о типах данных. 3 Выбор типов данных. 4 Создание временных таблиц. 5 Определение связей между таблицами. 6 Определение внешних и родительских ключей таблиц. 7 Определение рационального состава индексов таблиц. 8 Создание индексов таблиц. 9 Принципы создания индексов. 10 Правила создания хорошего индекса.	Самостоятельное изучение			**2**			Защита отчета по лаб. работе
6.3	Установка связи с таблицами баз данных 1 Использование средств ODBC. 2 Описание ODBC-источников данных. 3 Создание источников данных ODBC. 4 Преимущества и недостатки использования ODBC. 5 Установка связи с данными на SQL-сервере. 6 Определение прав доступа к данным.	Самостоятельное изучение						
6.4	Создание и использование представлений в среде SQL-сервера 1 Создание представлений в среде SQL-сервера. 2 Типы представлений в среде SQL-сервера. 3 Использование представлений в среде SQL-сервера. 4 Изменение значений с помощью представлений в среде SQL-сервера. 5 Контроль добавления и изменения записей в соответствии с условиями запроса представления в среде SQL-сервера. 6 Уничтожение представлений в среде SQL-сервера. .	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
6.5	Хранимые процедуры 1 Назначение хранимых процедур. 2 Язык написания хранимых процедур. 3 Определение хранимых процедур. 4 Определение и использование параметров хранимых процедур. Вызов хранимых процедур из приложения. 5 Отображение и редактирование хранимых процедур. Изменение и удаление хранимых процедур. **6 Компиляция процедуры и преобразование имен.**	**2**			**2**			Защита отчета по лаб. работе
6.6	Создание и использование курсоров 1 Понятие курсора. 2 Клиентские и серверные курсоры. 3 Использование курсоров SQL-сервера: объявление курсора, открытие курсора, выборка данных из курсора, закрытие курсора, аннулирование курсора. 4 Примеры использования курсоров. 5 Использование вложенных курсоров. 6 Работа с курсорами в клиентских приложениях для организации циклов обработки результатов запросов.	Самостоятельное изучение						
6.7	Триггеры SQL-сервера 1 Понятие о триггерах SQL-сервера. 2 Назначение триггеров: поддержка ссылочной целостности базы данных, ведение журналов изменений базы данных, реализация функций контроля за корректностью данных, накопление итоговой информации. 3 Виды триггеров по типам операций с базой данных. 4 Виды триггеров по обработке отдельных строк или запросов. Ограничения, налагаемые триггерами	**2**						
6.8	Использование триггеров SQL-сервера (1 Создание триггеров. 2 Удаление триггеров. 3 Отображение информации о триггере. 4 Использование триггеров для операторов Insert и Update. 5 Использование триггеров для оператора Delete. 6 Использование вложенных триггеров.	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
6.9	Транзакции (4 ч.) 1 Понятие транзакций. 2 Понятие блокировки. 3 Откат транзакций в триггерах. 4 Специальные методы управления транзакциями. 5 Реализация триггеров в процессе создания клиент-серверного приложения баз данных.	Самостоятельное изучение						
6.10	Основные понятия MySQL **1. Структура языка SQL принятого в MySQL.** **2. Основные понятия MySQL.** **3. Примеры синтаксиса.** **4. Типы столбцов в таблицах.** **5. Манипуляции с данными в таблицах** **6. Проблемы защиты и система привилегий пакета.**	Самостоятельное изучение						
6.11	**Администрирование сетевых БД** 1 Функции администратора БД. 2 Работа с пользователями, установление стадартов и процедур. 3 Задачи администратора БД. 4 Целостность БД. **5 Обработка транзакций, контроль параллельной обработки**.	Самостоятельное изучение						
6.12	**Защита сетевых БД** 1 Идентификация пользователя. 2 Проверка полномочий и представления данных, шифровка. 3 Восстановление БД, источники отказов, процедуры восстановления, протоколирование с отложенными обновлениями, протоколирование с немедленными обновлениями, контрольные точки.	Самостоятельное изучение						

1	2	3	4	5	6	7	8	9
6.13	**Совместное использование данных и базы данных)** 1 Совместное использование данных пользователями разных уровней. 2 Совместное использование данных региональными отделениями. 3 Критерии эффективности БД. 4 БД и контроль управления.	Самостоятельное изучение						
7	**БАЗЫ ДАННЫХ В ИНТЕРНЕТ**							
7.1	**Сервер Web как ядро приложений для Интернет** 1 Основы работы сервера Web. 2 Пассивные и активные серверы Web. 3 Активность на стороне клиента. 4 Интеграция серверов Web и SQL Server. **5** Проект Интернет магазина. Установка программ и подготовка к работе.	Самостоятельное изучение						
7.2	**Связь приложений с базами данных через ODBC** 1. Программный интерфейс ODBC. 2. Обработка ошибок. 3. Программа ODBCAPP. 4. Запуск хранимых процедур.	Самостоятельное изучение						
	Всего по дисциплине 5й семестр	**2**			**8**			экзамен
	Всего по дисциплине	**20**			**16**			

Старший преподаватель кафедры АСОИ В.Н.Леванцов

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ТЕМ ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Перечень лабораторных работ

1. Разработка структуры реляционной базы данных.
2. Создание и заполнение таблицы данных в интерактивном режиме
3. Создание схемы данных.
4. Проектирование форм и отчетов.
5. Использование запроса выборки данных для анализа отдельных таблиц.
6. Анализ данных связанных таблиц с помощью SQL.
7. Модификация базы и структур таблиц средствами SQL.
8. Разработка механизмов поиска информации.
9. Создание хранимых процедур.
10. Создание таблиц баз данных в среде SQL-сервера.
11. Создание и использование представлений в среде SQL-сервера.
12. Использование баз данных в сети Интернет.

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ НЕОБХОДИМОГО ОБОРУДОВАНИЯ И КОМПЬЮТЕРНЫХ ПРОГРАММ

1. Класс современных программных ЭВМ.
2. СУБД Microsoft Access.
3. Сетевая СУБД класса MySQL.
4. Сетевая СУБД класса Microsoft SQL Server.
5. Сетевая СУБД класса DB2.

РЕКОМЕНДАЦИИ ПО ОРГАНИЗАЦИИ И ВЫПОЛНЕНИЮ УСР

Для самостоятельного изучения выделяются следующие темы:
- СУБД MS Access. Работа в интерактивном режиме;
- проблема аномалии и задача нормализации данных;
- описание структуры базы данных;
- ссылочная целостность данных;
- создание таблиц баз данных в среде SQL-сервера;
- создание и использование курсоров;
-основные понятия MySQL;
-защита сетевых БД;
-совместное использование данных и базы данных.

Тема 2.2 СУБД MS Access. Работа в интерактивном режиме – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию в умении описывать историю вопроса.

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие знания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.
2. Исправьте ошибки в определениях.
3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты, лабораторная работа.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.
2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.
3. Объясните принципы определения интерактивного режима.

Форма выполнения заданий – индивидуальная.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Опишите принципы реляционных СУБД.
2. Приведите классификацию СУБД.
3. Опишите организацию работы в интерактивном режиме.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – реферат.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.
2) Конспект лекций по дисциплине.
3) Информация в сети Интернет.

Тема 3.1 Проблема аномалии и задача нормализации данных – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию в области проектирования БД

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие задания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.
2. Исправьте ошибки в определениях.
3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.

2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.

3. Дайте определение 1, 2 и 3 нормальной формах.

Форма выполнения заданий – индивидуальная.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Приведите порядок и этапы проектирования БД.

2. Приведите примеры связей между объектами в БД.

3. Опишите дополнительные рекомендации по проектированию БД.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – реферат.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.

2) Конспект лекций по дисциплине.

3) Информация в сети Интернет.

Тема 4.3 Описание структуры базы данных – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию по описанию структуры БД.

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие задания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.

2. Исправьте ошибки в определениях.

3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.

2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.

3. Объясните принципы работы методов доступа к среде.

Форма выполнения заданий – тесты.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Опишите принципы работы систем быстрой разработки приложений.

2. Приведите примеры современных RAD- систем.

3. Продемонстрируйте порядок использования вычисляемых полей.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – практическая работа.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.

2) Конспект лекций по дисциплине.

3) Информация в сети Интернет.

Тема 5.2 Ссылочная целостность данных – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию в умении управления данными.

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие задания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.
2. Исправьте ошибки в определениях.
3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.
2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.
3. Объясните принципы ссылочной целостности данных.

Форма выполнения заданий – тесты.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Опишите методы обеспечения целостности данных.
2. Приведите примеры определение внешних ключей таблиц.
3. Продемонстрируйте приемы способов задания каскадных обновлений таблиц.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – практическая работа, лабораторная работа.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.
2) Конспект лекций по дисциплине.
3) Информация в сети Интернет.

Тема 6.2 Создание таблиц баз данных в среде SQL-сервера – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию в умении создания таблиц баз данных в среде SQL-сервера.

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие задания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.
2. Исправьте ошибки в определениях.
3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.

2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.

3. Объясните принципы работы в среде SQL-сервера.

Форма выполнения заданий – тесты.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Приведите основные способы создания таблиц баз данных в среде SQL-сервера.

2. Приведите основные типы данных в среде SQL-сервера.

3. Продемонстрируйте принципы определение внешних и родительских ключей таблиц.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – практическая работа, лабораторная работа.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.

2) Конспект лекций по дисциплине.

3) Информация в сети Интернет.

Тема 6.6 Создание и использование курсоров – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию в умении управлять курсорами в сетевой среды.

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие задания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.

2. Исправьте ошибки в определениях.

3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.

2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.

3. Объясните принципы работы клиентских и серверных курсоров.

Форма выполнения заданий – тесты.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Опишите порядок работы объявление курсора, открытие курсора, выборка данных из курсора, закрытие курсора, аннулирование курсора.

2. Приведите примеры использования курсоров.

3. Продемонстрируйте принципы использования вложенных курсоров.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – практическая работа, лабораторная работа.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.

2) Конспект лекций по дисциплине.

3) Информация в сети Интернет.

Тема 6.10 Основные понятия MySQL – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию в умении инсталляции и работы в среде MySQL

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие задания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.

2. Исправьте ошибки в определениях.

3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.

2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.

3. Приведите структура языка SQL принятого в MySQL.

Форма выполнения заданий – тесты.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Опишите примеры синтаксиса в MySQL.

2. Приведите примеры манипуляции с данными в таблицах.

3. Продемонстрируйте принципы защиты и система привилегий пакета в MySQL.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – практическая работа, лабораторная работа.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.

2) Конспект лекций по дисциплине.

3) Информация в сети Интернет.

Тема 6.12 Защита сетевых БД – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию в умении защиты данных в сетевой среде.

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие задания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.
2. Исправьте ошибки в определениях.
3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.
2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.
3. Приведите примеры проверки полномочий и представления данных, и их шифровки.

Форма выполнения заданий – тесты.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Опишите порядок восстановление БД и источники отказов.
2. Приведите примеры протоколирование с отложенными обновлениями.
3. Продемонстрируйте принципы работы контрольных точек.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – практическая работа, лабораторная работа.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.
2) Конспект лекций по дисциплине.
3) Информация в сети Интернет.

Тема 6.13 Совместное использование данных и базы данных – 2 часа

Цели: 1) овладеть знаниями по данной теме, терминологией и методологией; 2) сформировать компетенцию в умении совместного использования данных пользователями разных уровней.

Виды заданий УСР по теме с учетом модулей сложности:

А) Задания, формирующие задания по учебному материалу на уровне узнавания:

1. Соотнесите термины с определениями.

2. Исправьте ошибки в определениях.

3. Вставьте в определение соответствующий термин.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – тесты.

Б) Задания, формирующие компетенции на уровне воспроизведения:

1. Дайте определения терминам.

2. Приведите примеры, подтверждающие или опровергающие правильность утверждений.

3. Приведите примеры совместного использование данных региональными отделениями

Форма выполнения заданий – тесты.

Форма контроля выполнения заданий – тесты, контрольные вопросы.

В) Задания, формирующие компетенции на уровне применения полученных знаний:

1. Опишите порядок восстановление БД и источники отказов.

2. Приведите критерии эффективности БД.

3. Продемонстрируйте принципы работы БД и контроль управления.

Форма выполнения заданий - индивидуальная.

Форма контроля выполнения заданий – практическая работа, лабораторная работа.

Учебно-методическое обеспечение:

1) Рекомендуемая основная и дополнительная литература.

2) Конспект лекций по дисциплине.

3) Информация в сети Интернет.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

ОСНОВНАЯ

1. Голицына, О.Л. Базы данных: учебное пособие / О.Л. Голицына, Н.В. Максимов, И.И. Попов. – Москва: ИНФРА-М, 2014. – 400 с.
2. Змитрович, А.И. Базы данных и знаний : учебное пособие / А.И. Змитрович, В.В. Апанасович, В.В. Скакун. – Минск : БГУ, 2007. – 363 с.
3. Кузин, А.В. Базы данных : учебное пособие для студентов высших учебных заведений, обучающихся по направлению подготовки "Информатика и вычислительная техника" / А.В. Кузин, С.В. Левонисова. – 6-е изд., стереотипное. – Москва : Академия, 2016. – 314 с.
4. Шустова, Л.И. Базы данных : учебник для студентов вузов по направлению "Прикладная информатика" / Л.И. Шустова, О.В. Тараканов. – Москва: ИНФРА-М, 2016. – 304 с.

ДОПОЛНИТЕЛЬНАЯ

1. Боуман, Д. Практическое руководство по SQL / Д. Боуман, С. Эмерсон, М. Дарновски. – Санкт-Петербург : Вильямс, 2001. – 352 с.
2. Вьейра, Р. SQL Server 2000. Программирование : в 2 ч. / Р. Вьейра. – Москва : Бином, 2004.
3. Бернс, П. Секреты Access для Windows / П. Бернс, Д. Николсон. – Киев, Диалектика, 1996. – 560 с.
4. Винкоп, С. Использование Microsoft SQL Server 7.0 / С. Винкоп. – Специальное издание. – Санкт-Петербург : Вильямс, 2001. – 816 с.
5. Горев, А. Эффективная работа с СУБД / А. Горев, Р. Ахаян, С. Макашарипов. – Санкт-Петербург : Питер, 1997. – 704 с.
6. Кауфман, Д. SQL. Программирование / Д. Кауфман, Б. Матсик, К. Спенсер. – Москва : Бином, 2002. – 744 с.
7. Крёнке, Д. Теория и практика построения баз данных / Д. Крёнке. – 9-е изд. – Санкт-Петербург [и др.] : Питер : Питер принт, 2005. – 858 с.
8. Моррисон, Дж. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Дж. Моррисон, А. Ухтомский, М. Пере. – Москва : Вильямс, 2001. – 1120 с.
9. Проектирование и реализация баз данных Microsoft SQL Server 2000 : учебный курс MCAD/ MCSE, MCDBA : сертификац. экзамен 70-229: офиц. пособие Microsoft® для самостоят. подгот. – 2-е изд., испр. – Москва : Русская редакция, 2003. – 512 с.
10. Риордан, Р. Программирование в Microsoft SQL Server 2000. Шаг за шагом / Р. Риордан. – Москва : Эком, 2002. – 608 с.
11. Ульман, Д. Системы баз данных. Полный курс / Д. Ульман, Г. Гарсиа-Молина, Дж. Уидом. – Москва : Вильямс, 2003. – 1088 с.

12.Фрост, Р. Базы данных. Проектирование и разработка : самоучитель : [реляционная теория, концептуальное проектирование, продвинутое проектирование баз данных, разработка баз данных, работа с данными, разработка приложений трехъярусной архитектуры, диаграммы] / Р. Фрост, Д. Дей, К. Ван Спайк. – Москва : NT Press, 2007. – 590 с.
13.Фуфаев, Э.В.Базы данных : учебное пособие для студентов ссузов / Э.В. Фуфаев, Д.Э. Фуфаев. – 4-е изд., стер. – Москва : Академия, 2008. – 320 с.
14.Харитонова, И.А. ACCESS 2018 / И.А. Харитонова, В.Д. Михеева – Санкт-Петербург :БХВ-Петербург, 2019. – 832 с.
15.Хоффбауер, М. ACCESS 7.0: сотни полезных рецептов / М. Хоффбауер, К. Шпильманн. – Киев : BHV, 1996. – 400 с.

ЭЛЕКТРОННЫЕ РЕСУРСЫ

1 Свободная энциклопедия ВикипедиЯ [Электронный ресурс]. – 2022. – Режим доступа: http://ru.wikipedia.org. – Дата доступа: 12.03.2022.
2 Интернет университет информационных технологий [Электронный ресурс]. – 2022. – Режим доступа: http://www.intuit.ru. – Дата доступа: 12.03.2022.
3 Информационно-справочный портал технической информации Хабрахабр [Электронный ресурс]. – 2022. – Режим доступа: https://habr.com/ru/all/. – Дата доступа: 12.03.2022.
4 Информационно-справочный портал технической информации Xgu.ru [Электронный ресурс]. – 2022. – Режим доступа: http://xgu.ru/. – Дата доступа: 12.03.2022.

Библ. Инж И.В. Никишина

ПРОТОКОЛ СОГЛАСОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ ПО ИЗУЧАЕМОЙ УЧЕБНОЙ ДИСЦИПЛИНЕ С ДРУГИМИ ДИСЦИПЛИНАМИ СПЕЦИАЛЬНОСТИ

Название дисциплины, с которой требуется согласование	Название кафедры	Предложения об изменениях в содержании учебной программы по изучаемой учебной дисциплине	Решение, принятое кафедрой, разработавшей учебную программу (с указанием даты и номера протокола)
Объектно-ориентированное программирование	Кафедра АСОИ		Рекомендовать к утверждению учебную программу в представленном варианте протокол № 11 от 18.06.2021
ПСПК	Кафедра АСОИ		Рекомендовать к утверждению учебную программу в представленном варианте протокол № 11 от 18.06.2021
Информационные системы и технологии	Кафедра АСОИ		Рекомендовать к утверждению учебную программу в представленном варианте протокол № 11 от 18.06.2021

ДОПОЛНЕНИЯ И ИЗМЕНЕНИЯ К УЧЕБНОЙ ПРОГРАММЕ
ПО ИЗУЧАЕМОЙ УЧЕБНОЙ ДИСЦИПЛИНЕ
на _____/_____ учебный год

№ № пп	Дополнения и изменения	Основание

Учебная программа пересмотрена и одобрена на заседании кафедры АСОИ
(протокол № ____ от _____ 20___ г.)
Заведующий кафедрой АСОИ
к.т.н., доцент _____ А.В.Воруев

УТВЕРЖДАЮ
Декан факультета физики и ИТ
УО «ГГУ имени Ф. Скорины»
к.ф.-м.н., доцент _____ Д.Л. Коваленко