МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра биомедицинской инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
О.Г. Локтионова
«12» 09 2023 г.

МЕДИЦИНСКИЕ БАЗЫ ДАННЫХ И ЭКСПЕРТНЫЕ СИСТЕМЫ

Методические указания по выполнению практических работ
для студентов направления подготовки: 12.03.04 «Биотехнические системы и технологии»

Курск 2023

УДК 615.478

Составители: С.Н. Родионова. В.В. Аксёнов

Рецензент:
Кандидат технических наук, доцент *Т.Н. Конаныхина*

Медицинские базы данных и экспертные системы: методические указания по выполнению практических работ для студентов направления подготовки: 12.03.04 «Биотехнические системы и технологии» / Юго-Зап. гос. ун-т; сост.: С.Н. Родионова, В.В. Аксёнов. Курск, 2023. - 35 с.

Содержатся теоретические и справочные сведения, необходимые для создания медицинских баз данных и экспертных систем медицинского назначения. Приводятся сведения об уровнях представления данных и автоматизированных информационных системах, которые основаны на данных, описываются модели данных.

Методические указания соответствуют требованиям федеральных государственных образовательных стандартов.

Предназначены для студентов для студентов направления подготовки: 12.03.04 «Биотехнические системы и технологии»

.

Текст печатается в авторской редакции

Подписано в печать . Формат 60х84 1/16.
Усл.печ. л. __. Уч.-изд. л. __. Тираж 30 экз. Заказ *888* .Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

ПРАКТИЧЕСКАЯ РАБОТА №1. ПРЕДМЕТНАЯ ОБЛАСТЬ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Цель работы: Изучить особенности задания «предметной области» информационной системы.

Теоретическая часть:

Предметная область - это часть реального мира, данные о которой мы хотим отразить в базе данных (например, бухгалтерия какого-либо предприятия, отдел кадров, банк, магазин и т.д.) Предметная область бесконечна и может содержать как существенно важные понятия и данные, так и малозначащие или вообще не значащие данные. Предметная область (ПрО) информационной системы рассматривается как совокупность реальных процессов и объектов (сущностей), представляющих интерес для её пользователей [6]. Каждая из сущностей предметной области обладает определённым набором свойств (атрибутов).

Для упрощения процедуры описания предметной области в большинстве случаев прибегают к определению типов сущностей, которые позволяют выделить из всего множества сущностей предметной области группу сущностей, однородных по структуре и поведению. Так, например, для *предметной области* "Больница" в качестве типов сущностей могут рассматриваться пациенты, врачи, отделения и т.п.

Данные предметной области представляются *экземплярами сущностей* (врач Петров, пациент Иванов, отделение функциональной диагностики). Экземпляры сущностей одного типа обладают одинаковыми наборами атрибутов, но должны отличаться значением хотя бы одного атрибута для того, чтобы быть узнаваемыми (например, студенты могут иметь одинаковые ФИО, но должны иметь разные номера зачётных книжек).

Среди атрибутов сущности можно выделить существенные и малозначительные. Признание какого-либо свойства существенным носит относительный характер. Например, атрибут *Должность* для сотрудника является существенным, а для читателя библиотеки - малозначительным.

Атрибуты можно условно классифицировать следующим образом:

– *Идентифицирующие и описательные атрибуты.* Идентифицирующие атрибуты имеют уникальное значение для

сущностей данного типа, описательные заключают в себе интересующие свойства сущности.

– *Составные и простые атрибуты.* Простой атрибут состоит из одного компонента, его значение неделимо; составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных.

– *Однозначные и многозначные атрибуты* (могут иметь соответственно одно или много значений для каждого экземпляра сущности).

– *Основные и производные атрибуты.* Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов.

– *Обязательные и необязательные.* Значение обязательного атрибута всегда устанавливается при помещении данных в БД; значение необязательного атрибута может быть пропущено.

Спецификация атрибута состоит из его названия, типа данных, размера и описания ограничений целостности - множества значений, которые может принимать данный атрибут.

Практическая часть.

Задание 1. Выделите базовые и зависимые сущности для различных ПрО:

Вариант 1: «Стоматологическая клиника», «Отдел кадров», «Магазин канцтоваров», «Институт»;

Вариант 2: «Библиотека», «Юридическая фирма», «Школа», «Театр»;

Вариант 3: «Автосервис», «Офтальмологическая клиника», «Нотариальная контора», «Туристическая фирма».

Задание 2. Определить набор атрибутов для различных сущностей заданных предметных областей.

Контрольные вопросы.

1. Дайте определение базы данных.
2. Дайте определение автоматизированной информационной системы.
3. Что такое предметная область базы данных?
4. Что такое экземпляр сущности? Приведите примеры.
5. Что такое атрибуты?
6. Как классифицируются атрибуты? Приведите примеры.

ПРАКТИЧЕСКАЯ РАБОТА №2. ПОСТРОЕНИЕ СВЯЗЕЙ МЕЖДУ СУЩНОСТЯМИ.

Цель работы: Изучить особенности построения связей между сущностями различных предметных областей.

Теоретическая часть:

Между сущностями ПрО могут существовать связи, имеющие различную семантику (содержательный смысл). Например, студент *учится* в группе, врач *лечит* пациента, клиент *имеет* вклад в банке. Связи могут быть факультативными или обязательными. Если вновь порождённая сущность одного из типов оказывается по необходимости связанной с сущностью другого типа, то между этими типами сущностей есть обязательная связь. Иначе связь является факультативной. Примеры обязательной и факультативной связей приведены на рис. 2.1. Здесь связь *замещает* является обязательной (изображается двойной линией), потому что каждый сотрудник должен работать на определённой должности, а связь *замещается* является факультативной, т.к. должность может быть вакантна (наименования сущностей, атрибутов и связей выделяются курсивом и подчёркиванием, сущность записывается прописными буквами *(ОТДЕЛ),* атрибут сущности начинается с прописной буквы *(Название)*, ключевой атрибут выделяется полужирным шрифтом *(Табельный номер)*, связь между сущностями определяется глаголом *(работает)*.)

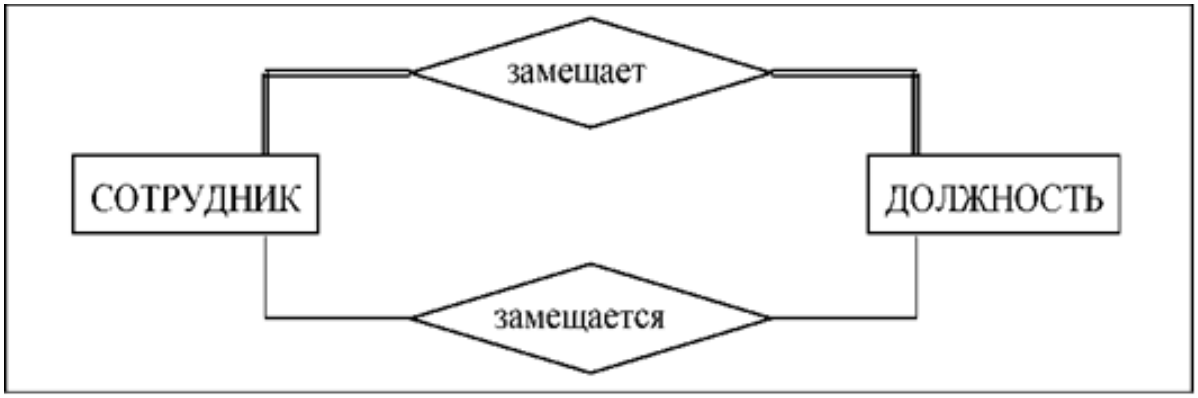

Рисунок 2.1. Примеры обязательной и факультативной связей.

Для удобства каждую связь между сущностями можно изображать одним ромбом (рис. 2.2). Выделяют также показатель кардинальности связи: "один к одному" (1:1), "один ко многим" (1:n) и "многие ко многим" (m:n) (рис. 2.2).

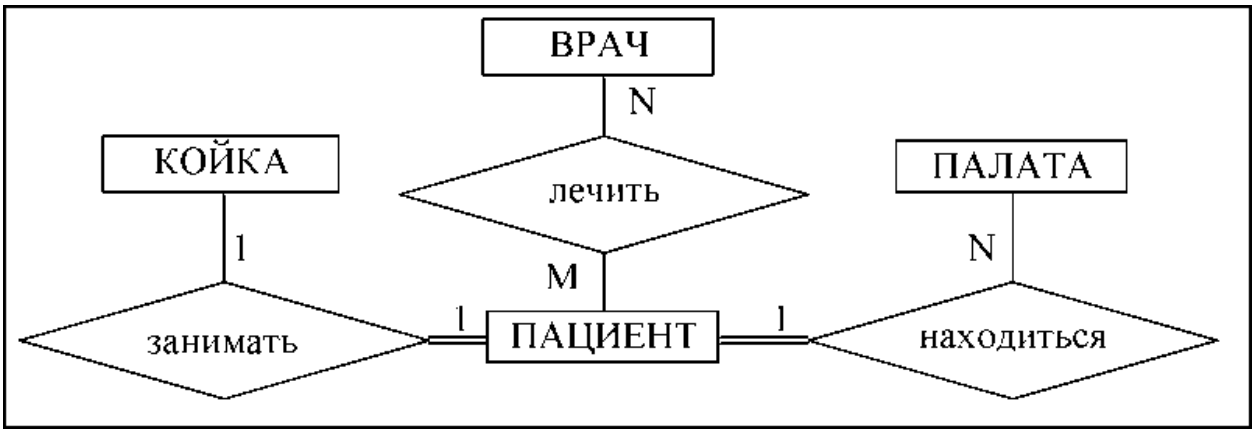

Рисунок 2.2. Примеры различной кардинальности связей

Связи, приведённые на рис. 2.2, с учётом семантики означают следующее:

- пациент-койка (1:1) - каждый пациент занимает одну койку, каждая койка в каждый момент времени может быть занята только одним пациентом;
- палата-пациент (1:n) - каждый пациент находится в одной палате, в каждой палате могут находиться несколько пациентов;
- пациент-врач (n:m) - каждый пациент может лечиться у нескольких врачей, каждый врач может лечить несколько пациентов.

Обратите внимание: необязательная связь имеет модификатор "может", а у обязательной связи его нет.

Степень связи - это количество типов сущностей, которые входят в связь. Различают унарные (рис. 2.3,а), бинарные (рис. 2.3,б) и тернарные (рис. 2.3,в) связи. (На практике связи с большей степенью редко используются). Унарная связь означает, что одни экземпляры сущности связаны с другими экземплярами этой же сущности (например, одни сотрудники руководят другими, а деталь может являться частью механизма).

Различают *тип связи* и *экземпляр связи.* Тип связи определяется её именем, обязательностью, степенью и кардинальностью, например, бинарная связь *учится* между сущностями *ГРУППА* и *СТУДЕНТ,* обязательная для студента, кардинальностью 1: n. А экземпляр связи - это конкретная связь между студентом Сидоровым и группой Б-11, в которой он учится.

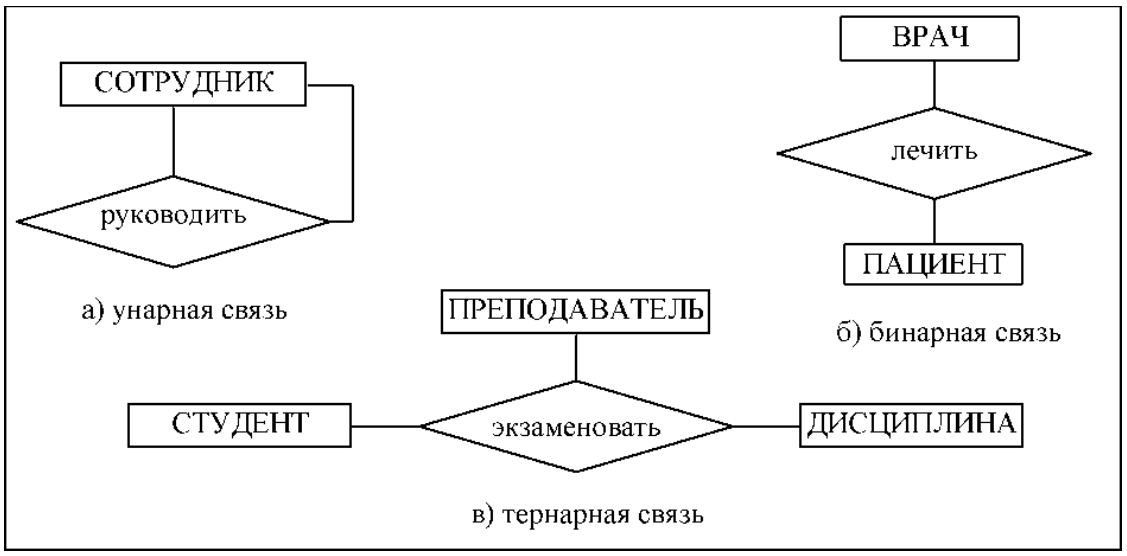

Рисунок. 2.3. Примеры связей различной степени

Совокупность типов сущностей и типов связей между ними характеризует структуру предметной области. Собственно, данные представлены экземплярами сущностей и связей между ними. Данные экземпляров сущностей и связей хранятся в базе данных информационной системы, а описание типов сущностей и связей является метаданными.

Множества экземпляров сущностей, значения атрибутов сущностей и экземпляры связей между ними могут изменяться во времени. Поэтому каждому моменту времени можно сопоставить некоторое состояние предметной области. Состояния ПрО должны подчиняться совокупности правил, которые характеризуют семантику предметной области. В базе данных эти правила могут быть заданы с помощью так называемых *ограничений целостности,* которые накладываются на атрибуты сущностей, типы сущностей, типы связей и/или их экземпляры. Фактически ограничения целостности - это правила, которым должны удовлетворять значения данных в БД. Например, для библиотеки можно привести такие ограничения целостности: количество экземпляров книги не может быть отрицательным; номер паспорта читателя должен быть уникальным; каждая книга относится к определённому разделу рубрикатора ББК - библиотечно-библиографической классификации и т.д.

Для того чтобы обеспечить соответствие базы данных текущему состоянию предметной области, база данных *динамически обновляется* (периодически или в режиме реального времени). Это обновление называется актуализацией данных. Актуализация может проводиться:

- вручную, если изменения в данные вносит пользователь (например, запись сведений о выдаче абоненту книги в библиотеке);
- автоматизировано, если изменения инициируются пользователем, но выполняются программно (например, обновление списка должников в библиотеке - читателей, которые просрочили дату возврата книг);
- автоматически, если данные поступают в электронном виде и обрабатываются программой без участия человека (это касается, например, автоматизированных систем управления производством).

Правильность обновлений может контролироваться программно, но правильнее контролировать их автоматически с помощью ограничений целостности БД.

База данных является информационной моделью внешнего мира, некоторой предметной области. Во внешнем мире сущности ПрО взаимосвязаны, поэтому в БД эти связи должны быть отражены. Если связи между данными в БД отсутствуют, то имеет смысл говорить о нескольких независимых БД и хранить их раздельно.

Практическая часть.

Задание 1. Определить обязательные и факультативные связи между сущностями в заданных ПрО:

Вариант 1: «Стоматологическая клиника», «Отдел кадров», «Магазин канцтоваров», «Институт»;

Вариант 2: «Библиотека», «Юридическая фирма», «Школа», «Театр»;

Вариант 3: «Автосервис», «Офтальмологическая клиника», «Нотариальная контора», «Туристическая фирма».

Задание 2. Приведите примеры связей различной степени (кардинальности) для ПрО из задания 1.

Контрольные вопросы.

1. Какие связи существуют между сущностями ПрО?
2. Что подразумевают под типом связи?
3. Как классифицируются атрибуты сущностей?
4. Какие показатели кардинальности связей выделяют? Приведите примеры.
5. Что подразумевают под экземпляром связи?
6. Перечислите способы актуализации данных?

ПРАКТИЧЕСКАЯ РАБОТА №3. ПОСТРОЕНИЕ СЕТЕВОЙ И ИЕРАРХЧЕСКОЙ МОДЕЛЕЙ ДАННЫХ

Цель работы: Изучить архитектуру сетевых и иерархических моделей данных.

Теоретическая часть:

Сетевая модель данных (СМД) – логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных (рис. 3.1).

Рисунок 3.1. Пример сетевой модели данных «Город»

Каждая вершина графа хранит экземпляры сущностей (записи одного типа) и сведения о групповых отношениях с сущностями других типов. Каждая запись может хранить произвольное количество значений атрибутов (элементов данных и агрегатов), характеризующих экземпляр сущности. Для каждого типа записи выделяется первичный ключ - атрибут, значение которого позволяет однозначно идентифицировать запись среди экземпляров записей данного типа.

Связи между записями в СМД выполняются в виде указателей, т.е. каждая запись хранит ссылку на другую однотипную запись (или признак конца списка) и ссылки на списки подчинённых записей, связанных с ней групповыми отношениями. Таким образом, в каждой вершине записи хранятся в виде связного списка. Если список организован как однонаправленный, запись имеет ссылку на следующую однотипную запись в списке; если список двунаправленный - то на следующую и предыдущую однотипные записи.

Рассмотрим фрагмент еще одной БД "Предприятие" (рис. 3.2). Здесь записи типов *ОТДЕЛЫ* и *ОРГАНИЗАЦИИ-ЗАКАЗЧИКИ* являются владельцами записей типа *ПРОЕКТЫ* и они связаны групповыми отношениями соответственно *выполняют* и *заказывают*. Записи типов *ОТДЕЛЫ* и *ПРОЕКТЫ* являются владельцами записей типа *СОТРУДНИКИ* и они связаны групповыми отношениями *работают* и *выполняются*. Записи типа *СОТР УДНИКИ* являются владельцами записей типа *ДЕТИ*.

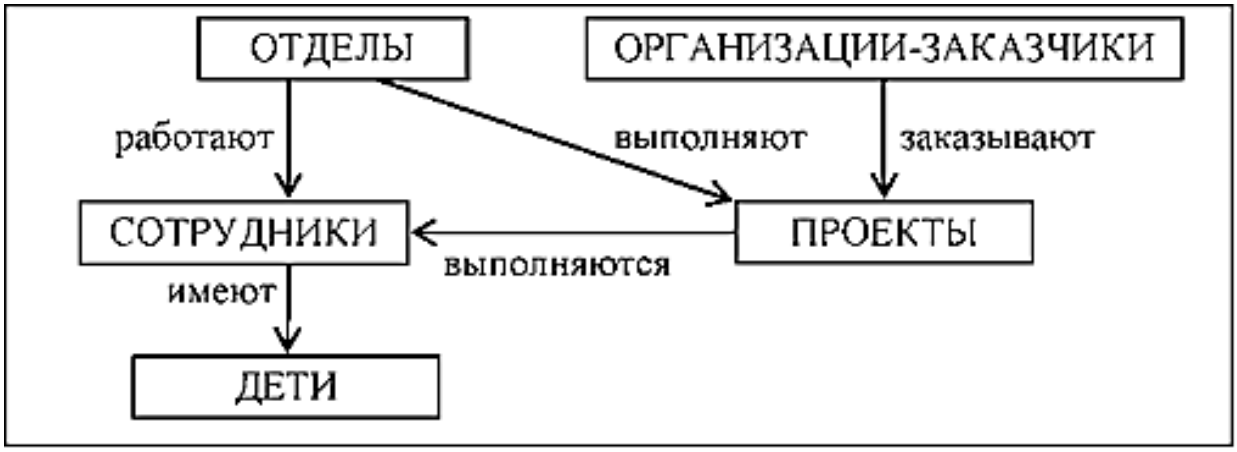

Рисунок 3.2. Пример фрагмента сетевой БД «Предприятие»

Групповые отношения чаще всего описывают связь "один-ко-многим": один владелец, много подчинённых. Например, отношение работают подразумевает, что каждый сотрудник работает в одном отделе, но в каждом отделе могут работать несколько сотрудников. С другой стороны, групповое отношение выполняются отражает связь "многие-ко-многим": каждый сотрудник может участвовать в выполнении нескольких проектов, каждый проект могут выполнять несколько человек. Что касается классов членства подчинённых записей, то связь "сотрудники-дети" относится к фиксированному классу членства, связь "сотрудники-проекты" - к необязательному, а все остальные - к обязательному классу членства. Режим включения для связи "сотрудники-проекты" ручной, для всех остальных - автоматический.

В СМД связи 1:n между разными сущностями реализуются с помощью групповых отношений, а связи 1:n между атрибутами сущности - в рамках записи. Для реализации связей типа n:m вводится вспомогательный тип записи и две связи 1: n.

Иерархическая модель данных (ИМД) позволяет строить БД с иерархической древовидной структурой. В основе ИМД лежит

понятие дерева. Дерево - это связный неориентированный граф, который не содержит циклов. При работе с деревом выделяют какую-то конкретную вершину, определяют её как корень дерева и рассматривают особо - в эту вершину не заходит ни одно ребро. В этом случае дерево становится ориентированным, ориентация определяется от корня. Дерево как ориентированный граф определяется так:

- имеется единственная особая вершина, называемая корнем, в которую не заходит ни одно ребро;
- во все остальные вершины заходит только одно ребро, а исходит произвольное количество ребер;
- граф не содержит циклов.

Конечные вершины, то есть вершины, из которых не выходит ни одной дуги, называются листьями дерева. Количество вершин на пути от корня к листьям в разных ветвях дерева может быть различным.

В иерархической модели данных используется ориентация древовидной структуры от корня к листьям. Графическая диаграмма концептуальной схемы базы данных называется деревом определения. Пример иерархической базы данных приведён на рис. 3.3. Каждая некорневая вершина в ИМД связана с родительской вершиной (сегментом) иерархическим групповым отношением. Каждая вершина дерева соответствует типу сущности ПрО. Тип сущности характеризуется произвольным количеством атрибутов, связанных с ней отношением 1:1. Атрибуты, связанные с сущностью отношением 1:n, образуют отдельную сущность (сегмент) и переносятся на следующий уровень иерархии. Реализация связей типа n:m не поддерживается.

Рисунок 3.3. Пример фрагмента иерархической БД «Университет»

Тип вершины определяется типом сущности и набором её атрибутов. Каждая вершина дерева хранит экземпляры сущностей - записи. Следствием внутренних ограничений иерархической модели является то, что каждому экземпляру зависимой группы в БД соответствует уникальное множество экземпляров родительских записей - по одному экземпляру (записи) каждого типа вершин вышестоящих уровней.

По сравнению с СМД иерархическая имеет ограниченный набор режимов включения и исключения подчинённых записей. Это определяется обязательностью связей: в дереве не может быть «висячих» вершин, не связанных с вершиной верхнего уровня (кроме корневой). Поэтому ИМД не поддерживает необязательный класс членства и ручной режим включения записей.

Практическая часть.

Задание 1. Построить сетевую модель данных для ПрО: «Стоматологическая клиника», «Магазин канцтоваров», «Библиотека» и объясните принцип формирования своей в ней,

Задание 2. Построить иерархическую модель данных для ПрО: «Стоматологическая клиника», «Магазин канцтоваров», «Библиотека» и объясните принцип формирования своей в ней,

Задание 3. Привести примеры, для каких задач лучше использовать иерархическую модель БД, а для каких сетевую при решении заданий 1 и 2 данной практической работы.

Контрольные вопросы.

1. Что такое сетевая модель БД?
2. Как организуются связи в сетевой модели БД?
3. Какими признаками характеризуются групповые отношения?
4. Какие операции над данными применяются в СМД?
5. Какие переходы возможны в СМД?
6. Что такое иерархическая модель БД?
7. Как организуются связи в иерархической модели БД?

8. Какие способы навигации предусмотрены в иерархической модели?

9. В чем отличия сетевой и иерархической моделей?

ПРАКТИЧЕСКАЯ РАБОТА №4. СОСТАВЛЕНИЕ РЕЛЯЦИОННЫХ ОТНОШЕНИЙ

Цель работы: Научиться составлять реляционные отношения для различных предметных областей, задавать первичные и внешние ключи для созданных отношений.

Теоретическая часть:

Базовой структурой РМД является отношение, основанное на декартовом произведении доменов. Домен - это множество значений, которое может принимать элемент данных (например, множество целых чисел, множество дат, множество комбинаций символов длиной N и т.п.). Домен может задаваться перечислением элементов, указанием диапазона значений, функцией и т.д.

Пусть D_1, D_2 ,..., D_k - произвольные конечные и не обязательно различные множества (домены). Декартово произведение этих множеств определяется следующим образом:

$D1xD2x...xD_k = \{(d_1, d2,..., dk) \mid d_i \in D_i, i=1,...,k\}$.

Таким образом, декартово произведение позволяет получить все возможные комбинации значений элементов исходных множеств.

Пример. Для доменов D_1 = (1, 2), D_2 = (A, B, C) декартово произведение $D = D_1xD_2$ будет таким: D = {(1,A), (1,B), (1,C), (2,A), (2,B), (2,C)}.

Подмножество декартова произведения доменов называется отношением.

Отношение содержит данные о сущностях определённого типа. Поясним это на примере. Если построить произведение трёх доменов *Должности* ('директор', 'бухгалтер', 'водитель', 'продавец'), *Оклады* (x | 20000<x<80000), *Надбавки* (1.1, 1.2, 1.3), то мы получим 4*60001*3=720012 комбинаций. Но реально отношение «Штатное расписание» содержит по одной строке на каждую должность, т.е. является именно подмножеством декартова произведения доменов.

Элементы отношения называют *кортежами* (или *записями*). Каждый кортеж отношения соответствует одному экземпляру сущности определённого типа. Элементы кортежа принято называть *атрибутами* (или *полями).*

Отношение обладает двумя основными свойствами:

1. В отношении не должно быть одинаковых кортежей, т.к. это множество.
2. Порядок кортежей в отношении несущественен.

Таким образом, в отношении не бывает первого, второго или последнего кортежа: при выводе данных отношения кортежи выводятся в произвольном порядке, если не задано упорядочение по значениям полей.

Отношение удобно представлять как таблицу, где строка является кортежем, а столбец соответствует домену (рис. 4.1, отношение *СТУДЕНТЫ).* Количество строк в таблице (кортежей в отношении) называется мощностью отношения, количество столбцов (атрибутов) - арностью.

домен 1	домен 2	домен 3 (ключ)	домен 4	домен 5
Группа	*ФИО студента*	*Номер зачётной книжки*	*Год рождени*	*Размер стипенди*
С-72	Волкова Елена Павловна	С-12298	1991	2900.00
С-91	Белов Сергей Юрьевич	С-12299	1990	2400.00
• • •				
С-72	Фролов Юрий Вадимович	С-14407	1991	0

Рисунок 4.1. Пример табличной формы представления отношения

Отношение имеет имя, которое отличает его от имён всех других отношений. Атрибутам реляционного отношения назначаются имена, уникальные в рамках отношения. Обращение к отношению происходит по его имени, а обращение к атрибуту - по имени отношения и имени атрибута.

Каждый атрибут определён на некотором домене, несколько атрибутов отношения могут быть определены на одном и том же домене (например, номера рабочего и домашнего телефонов). Домен задаётся типом данных, размером и ограничениями целостности: например, пол - это символьное поле длиной 1, которое может принимать значения из множества ('м', 'ж'). В реляционных базах данных поддерживаются такие типы данных как символьный, числовой, дата и некоторые другие (конкретный перечень типов зависит от СУБД).

Атрибут может быть обязательным и необязательным. Значение обязательного атрибута должно быть определено в момент внесения

данных в БД. Если атрибут необязательный, то для таких случаев предусмотрено специальное значение - NULL, которое можно интерпретировать как "неизвестное значение". Значение NULL не привязано к определённому типу данных, т.е. может назначаться данным любых типов.

Перечень атрибутов отношения с их типами данных и размерами определяют схему отношения. Отношения, построенные по одинаковой схеме, называют односхемными; по различным схемам - разносхемными.

Ключ отношения - это атрибут (группа атрибутов), значения которого классифицируют или идентифицируют кортеж. Например, значение атрибута *Группа* отношения *СТУДЕНТЫ* позволяет выделить среди всех студентов института студентов конкретной группы. Если ключ состоит из нескольких атрибутов, он называется *составным.* Если значения ключа уникальны в рамках столбца отношения, то такой ключ называется *потенциальным.* Потенциальных ключей может быть несколько (или не быть ни одного), но для отношения выделяется один основной ключ - первичный. Первичный ключ идентифицирует экземпляр сущности, его значение должно быть уникальным *(unique)* и обязательным *(not null).* (На рис. 4.1 первичный ключ выделен полужирным шрифтом). Неуникальные ключи ещё называют *вторичными.*

РМД не поддерживает групповые отношения (по версии CODASYL). Для связей между отношениями используются внешние ключи. Внешний ключ (*foreign key)* - это атрибут подчинённого (дочернего) отношения, который является копией первичного *(primary key)* или уникального *(unique)* ключа родительского отношения. (Пример - отношение *ОЦЕНКИ,* связанное с отношением *СТУДЕНТЫ* внешним ключом *Номер зачётной книжки,* рис. 4.2).

Номер зачётной	*Дисциплина*	*Оценка*
С-12298	Программирование	5
С-12298	Дискретная	4
С-14407	Программирование	3

Рисунок 4.2. Связь отношений "Оценки" и "Студенты" по внешнему ключу

Если связь необязательная, то значение внешнего ключа может быть неопределённым *(null)*.

Фактически внешние ключи логически связывают экземпляры сущностей разных типов (родительской и подчинённой сущностей).

Внешний ключ - это ограничение целостности, в соответствии с которым множество значений внешнего ключа является подмножеством значений первичного или уникального ключа родительской таблицы.

Ограничение целостности по внешнему ключу проверяется в двух случаях:

- при добавлении записи в подчинённую таблицу СУБД проверяет, что в родительской таблице есть запись с таким же значением первичного ключа;
- при удалении записи из родительской таблицы СУБД проверяет, что в подчинённой таблице нет записей с таким же значением внешнего ключа.

Примечание: внешний ключ может ссылаться на первичный ключ этой же таблицы. Это позволяет описывать унарную связь - иерархию однотипных сущностей. Например, если в таблицу *СОТРУДНИКИ* добавить поле *Руководитель* и описать его как внешний ключ на эту же таблицу, то в этом поле будет храниться идентификатор руководителя данного сотрудника (рис. 4.3). Атрибут *Руководитель* является необязательным.

Табельный номер	*№ отдела*	*ФИО*	*Должность*	*Руководитель*
002	1	Сухов К.А.	директор	
034	1	Петрова К.В.	секретарь	002
988	2	Рюмин В.П.	начальник	002
909	2	Серова ТВ.	вед.	988

Рисунок 4.3. Внешний ключ "Руководитель", ссылающийся на первичный ключ этой же таблицы

Практическая часть.

Задание 1. Составить реляционные отношения для различных ПрО:

– "Отдел кадров": список должностей, список сотрудников, список подразделений;

– "Магазин": список товаров, список поставщиков, список поставок;

– "Проектная организация": список отделов, список сотрудников, список проектов.

Задание 2. Определить первичные ключи для созданных отношений.

Задание 3. Определить внешние ключи для созданных отношений.

Контрольные вопросы.

1. Какова базовая структура реляционной модели данных (РМД)?
2. Дайте понятие домена?
3. Что такое кортеж?
4. Перечислите свойства отношений РМД?
5. Назовите типы атрибутов РМД.
6. Что такое ключ отношений? Приведите примеры составного и потенциального ключа.
7. Что такое внешний ключ?
8. Какие операции над данными предусмотрены в РМД?

ПРАКТИЧЕСКАЯ РАБОТА №5. СОЗДАНИЕ ЗАПРОСОВ НА ЯЗЫКЕ SQL.

Цель работы: Изучить виды, а также структуру запросов в SQL.

Теоретическая часть:

SQL (Structured Query Language, или язык структурированных запросов) — это декларативный язык программирования (язык запросов), который используют для создания, обработки и хранения данных в реляционных БД. Другими словами, SQL — это язык запросов для управления реляционными базами данных. По синтаксису SQL-запросы максимально похожи на обычные предложения, например:

```
SELECT (Name, Age) FROM Clients WHERE Age > 20
```

Означает: «Выбрать Имя и Возраст из Таблицы с клиентами, где Возраст больше 20».

Виды запросов в SQL

В SQL четыре вида запросов — **DDL**, **DML**, **DCL** и **TCL**.

DDL, или **data definition language**, нужен, чтобы определять данные. Эти запросы позволяют настраивать базу данных — создавать с нуля и прописывать её структуру. Примеры DDL-запросов: **CREATE**, **DROP**, **RENAME**, **ALTER**.

DML, или **data manipulation language**, нужен, чтобы управлять данными в таблицах. Эти запросы помогают добавлять, обновлять, удалять и выбирать данные. Примеры DML-запросов: **SELECT**, **UPDATE**, **DELETE**, **INSERT**.

DCL, или **data control language**, нужен, чтобы выдавать или отзывать права доступа для пользователей. Примеры DCL-запросов: **GRANT**, **REVOKE**, **DENY**.

TCL, или **transaction control language**, нужен, чтобы управлять транзакциями. Это могут быть запросы, связанные с подтверждением или откатом изменений в базе данных. Примеры TCL-запросов: **COMMIT**, **ROLLBACK**, **BEGIN**.

Классический SQL-запрос состоит из шести самых популярных операторов: два из них обязательные, а другие четыре используются по обстоятельствам.

- **SELECT** — выбирает отдельные столбцы или всю таблицу целиком (обязательный);
- **FROM** — указывает из какой таблицы получить данные (обязательный);
- **WHERE** — условие, по которому SQL выбирает данные;
- **GROUP BY** — формирует столбец, по которому будут группироваться данные;
- **HAVING** — условие, по которому сгруппированные данные будут отфильтрованы;
- **ORDER BY** — столбец, по которому данные будут отсортированы;

Остановимся на них подробнее.

SELECT

Любая команда должна начинаться с ключевого слова — или действия, которое должно произойти. Например, выбрать строку, вставить новую, изменить старую или удалить таблицу целиком.

Одно из таких ключевых слов — SELECT. Оно выбирает отдельные столбцы или таблицу целиком, чтобы потом передать данные другим запросам на обработку.

В качестве примера выберем столбцы Name и Age из таблицы Clients:

```
SELECT (Name, Age) FROM Clients
```

На выходе будут все строки таблицы, принадлежащие столбцам Name и Age.

FROM

Эта часть ставится после SELECT и нужна затем, чтобы указать, из какой таблицы или источника данных приходит информация. Здесь прописывается имя таблицы, с которой мы хотим работать.

Например, ранее мы уже выбирали данные из таблицы Clients:

```
SELECT (Name, Age) FROM Clients
```

В SQL всё построено на таблицах. Поэтому, если нужно получить данные из другого места — указываем другую таблицу.

WHERE

Если нужно отфильтровать данные, используем слово WHERE. После него указывается условие, которому должны удовлетворять строки, чтобы они попали в результат выполнения запроса.

Например, этот запрос вернёт все строки из таблицы, где значения Age больше 20:

```
SELECT (Name, Age) FROM Clients WHERE Age > 20
```

GROUP BY

Этот оператор помогает сгруппировать данные по определённым столбцам. В результате получим новую таблицу, составленную на основе выбранных данных.

Например, сгруппируем результат предыдущего запроса по городам:

SELECT (Name, Age) FROM Clients WHERE Age > 20 GROUP BY City

Запрос вернёт клиентов старше 20 лет и сгруппирует их по городам. Главное — чтобы столбец City присутствовал в таблице.

HAVING

Нужен, чтобы собирать группы по определённым условиям. Его обычно используют в паре с GROUP BY, а по своей функциональности он похож на WHERE.

Например, укажем, чтобы в группы добавлялись только клиенты с суммой заказа от 1000 рублей:

```
SELECT (Name, PaymentAmount, Age)
FROM Clients
WHERE Age > 20
GROUP BY City
HAVING PaymentAmount > 1000
```

ORDER BY

Позволяет сортировать полученные строки по возрастанию или убыванию. Работает как с числами, так и с символами. В качестве параметра нужно указать столбец, по которому надо выполнить сортировку.

Если надо отсортировать клиентов по возрасту от младшего к старшему, добавляют команду ORDER BY Age. В таком случае будем иметь следующий программный код:

```
SELECT (Name, PaymentAmount, Age)
FROM Clients
WHERE Age > 20
GROUP BY City
HAVING PaymentAmount > 1000
ORDER BY Age
```

Чтобы отсортировать по убыванию, добавляют оператор DESC:

```
SELECT (Name, PaymentAmount, Age)
FROM Clients
```

```
WHERE Age > 20
GROUP BY City
HAVING PaymentAmount > 1000
ORDER BY Age DESC
```

Примеры SQL-запросов: создаём первую базу данных

В качестве примера будем наполнять базу данных с кошками, живущими в разных городах России.

CREATE DATABASE

Первым делом создаём базу данных. Делается это с помощью команды CREATE DATABASE:

```
CREATE DATABASE CatsCatsCats;
```

Запрос CREATE TABLE создаёт таблицу в базе данных. В общем виде команда выглядит так:

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype
);
```

Чтобы задать свои параметры таблицы, на месте table_name пишем название, а в скобках указываем названия колонок и типы данных, которые они будут содержать.

В SQL много типов данных. Наиболее популярные из них:

- INT — целое число;
- DATETIME — дата;
- VARCHAR — строка;
- FLOAT — десятичное число.

В нашей таблице используется два типа: строки (VARCHAR) и целые числа (INT):

```
CREATE TABLE CatsAndOwners (
CatID int(6) NOT NULL,
CatName varchar(255) NOT NULL,
CatAge int(6) NOT NULL,
CatColor varchar(255) NOT NULL,
CatOwnerName varchar(255) NOT NULL
);
```

В примере выше мы добавили пять столбцов: уникальный номер кота CatID, его имя CatName, возраст CatAge, цвет CatColor и имя владельца CatOwnerName. А ещё задали, чтобы ни одно из полей не было пустым — NOT NULL.

Цифры рядом с типами данных обозначают, сколько бит выделяется для поля. Например, varchar (255) значит, что строка может принимать размер от 0 до 255 бит.

Тогдп таблица будет иметь вид:

CatID	CatName	CatAge	CatColor	CatOwnerName

ALTER TABLE

Новые колонки можно добавлять с помощью команды ALTER TABLE. Добавим город проживания кота:

```
ALTER TABLE CatsAndOwners
ADD City varchar(255);
```

В запросе указываем, в какую таблицу хотим внести изменения, а затем с помощью ключевого слова ADD добавляем название столбца и его тип данных. Теперь таблица примет вид:

CatID	CatName	CatAge	CatColor	CatOwnerName	City

ALTER TABLE рассчитана не только на добавление новых колонок, но и на удаление и редактирование существующих.

INSERT

Позволяет добавить новую строку в таблицу. Для этого нужно указать, какие столбцы необходимо заполнить и передать значения для них с помощью команды VALUES. Добавим еще несколько котов:

```
INSERT INTO CatsAndOwners(CatID, CatName, CatAge, CatColor, CatOwnerName, City)
VALUES
(1, 'Мурка', 3, 'Чёрная', 'Дмитрий', 'Москва');
```

```
INSERT INTO CatsAndOwners(CatID, CatName, CatAge,
CatColor, CatOwnerName, City)
VALUES
(2, 'Белла', 7, 'Белая', 'Максим', 'Саратов');

INSERT INTO CatsAndOwners(CatID, CatName, CatAge,
CatColor, CatOwnerName, City)
VALUES
(3, 'Симба', 5, 'Рыжий', 'Екатерина', 'Санкт-Петербург');

INSERT INTO CatsAndOwners(CatID, CatName, CatAge,
CatColor, CatOwnerName, City)
VALUES
(4, 'Лео', 2, 'Полосатый', 'Александр', 'Екатеринбург');

INSERT INTO CatsAndOwners(CatID, CatName, CatAge,
CatColor, CatOwnerName, City)
VALUES
(5, 'Мася', 1, 'Серый', 'Анна', 'Москва');
```

Важно знать, что строки указываются в одинарных кавычках, а числа — без них. Несколько строк одной командой добавить нельзя! Таблица будет иметь вид:

CatID	CatName	CatAge	CatColor	CatOwner Name	City
1	Мурка	3	Чёрная	Дмитрий	Москва
2	Белла	7	Белая	Максим	Саратов
3	Симба	5	Рыжий	Екатерина	Санкт-Петербург
4	Лео	2	Полосатый	Александр	Екатеринбург
5	Мася	1	Серый	Анна	Москва

SELECT

Запрос позволят доставать данные из таблицы. Достанем из таблицы список котов и их владельцев:

```
SELECT CatName, CatOwnerName
FROM CatsAndOwners;
```

Результат:

CatName	CatOwnerName
Мурка	Дмитрий
Белла	Максим
Симба	Екатерина
Лео	Александр
Мася	Анна

Если нужно выбрать все столбцы из таблицы, после слова SELECT добавим символ *. В этом случае на выходе получим всю таблицу целиком.

```
SELECT *
FROM CatsAndOwners;
```

WHERE

Нужен, чтобы задавать условия для фильтрации строк. Например, можем выбрать только те, у которых значение CatAge больше 5:

```
SELECT CatName, CatAge
FROM CatsAndOwners
WHERE CatAge > 5
```

Результатом будет одна строка с двумя столбцами:

CatName	CatAge
Белла	7

AND, OR, BETWEEN

Оператор WHERE интересен тем, что внутри него можно указывать условия — причём сразу несколько. Делается это с помощью логических конструкций AND, OR и BETWEEN.

AND — это логическое «И». Оно означает, что должны выполняться оба условия запроса одновременно. Например, кошка должна быть чёрной «И» проживать в Москве.

```
SELECT CatName
FROM CatsAndOwners
WHERE CatColor = 'Чёрная'
    AND City = 'Москва'
```

Результат:

CatName
Мурка

OR — это логическое «ИЛИ». Оно означает, что должно выполниться или одно условие, или второе. Например, кошка должна быть «ИЛИ» старше пяти лет, «ИЛИ» быть чёрной.

```
SELECT CatName
FROM CatsAndOwners
WHERE CatAge > 5
    OR CatColor = 'Чёрная'
```

Результат:

CatName
Мурка
Белла

BETWEEN — это оператор, который выбирает все элементы внутри заданного диапазона. Например, можно запросить всех кошек в возрасте от двух до шести лет.

```
SELECT CatName, CatAge
FROM CatsAndOwners
WHERE CatAge BETWEEN 2 AND 6
```

Результат:

CatName	CatAge
Мурка	3
Симба	5
Лео	2

Все вышеуказанные операторы можно использовать одним пакетом:

```
SELECT CatName, CatAge
FROM CatsAndOwners
WHERE CatAge BETWEEN 2 AND 8
    AND (City = 'Саратов' OR City = 'Санкт-Петербург')
    OR CatName = 'Мурка'
```

Результат:

CatName	CatAge
Мурка	3
Белла	7
Симба	5

ORDER BY

Сортирует полученные строки в заданном столбце по убыванию или по возрастанию. Например, можем выбрать всех кошек и отсортировать их от самых старших к самым младшим:

```
SELECT CatName, CatAge
FROM CatsAndOwners
ORDER BY CatAge DESC
```

Результат:

CatName	CatAge
Белла	7
Симба	5
Мурка	3
Лео	2
Мася	1

Чтобы отсортировать записи по возрастанию, нужно просто убрать из запроса параметр DESC:

```
SELECT CatName, CatAge
FROM CatsAndOwners
ORDER BY CatAge
```

Результат:

CatName	CatAge
Мася	1
Лео	2
Мурка	3
Симба	5
Белла	7

GROUP BY

Выбранные строки можно сгруппировать по столбцам. Например, можем посмотреть, сколько кошек живёт в разных городах.

```
SELECT City, COUNT(*) AS CatCount
FROM CatsAndOwners
GROUP BY City;
```

В этом примере мы применили агрегатную функцию COUNT, которая посчитала количество строк в каждой группе.

Результат:

CatName	CatCount
Москва	2
Саратов	1
Санкт-Петербург	1
Екатеринбург	1

Также мы использовали оператор AS, чтобы задать название для новой колонки, в которую мы и собрали количество котов в разных городах.

LIMIT

Запрос позволяет ограничить количество строк в финальной выдаче. Например, можем указать, чтобы выводились только первые две строки из таблицы:

```
SELECT *
FROM CatsAndOwners
LIMIT 2;
```

Результат:

CatID	CatName	CatAge	CatColor	CatOwner Name	City
1	Мурка	3	Чёрная	Дмитрий	Москва
2	Белла	7	Белая	Максим	Саратов

UPDATE

Позволяет изменить данные в таблице. Допустим, кошка Симба сменила цвет шёрстки на пурпурный.

```
UPDATE CatsAndOwners
SET CatColor = 'Пурпурный'
WHERE CatName = 'Симба';
```

Всё просто: рядом с командой UPDATE пишем название таблицы, которую нужно обновить, затем рядом с SET указываем, какой именно столбец меняем и на какое значение, а в конце — определяем конкретную ячейку.

Результат:

CatID	CatName	CatAge	CatColor	CatOwner Name	City
1	Мурка	3	Чёрная	Дмитрий	Москва
2	Белла	7	Белая	Максим	Саратов
3	Симба	5	Пурпурный	Екатерина	Санкт-Петербург
4	Лео	2	Полосатый	Александр	Екатеринбург
5	Мася	1	Серый	Анна	Москва

DELETE

Удаляет строку. Например, можем удалить из таблицы всех кошек, которые живут в Саратове:

```
DELETE FROM CatsAndOwners
WHERE City = 'Саратов';
```

Результат:

CatID	CatName	CatAge	CatColor	CatOwner Name	City
1	Мурка	3	Чёрная	Дмитрий	Москва
3	Симба	5	Пурпурный	Екатерина	Санкт-Петербург
4	Лео	2	Полосатый	Александр	Екатеринбург
5	Мася	1	Серый	Анна	Москва

DROP COLUMN

Удаляет столбец. Например, можно удалить имена кошачьих хозяев:

```
ALTER TABLE CatsAndOwners
DROP COLUMN CatOwnerName;
```

Заметьте, что сначала нужно применить команду ALTER TABLE. Как мы помним, она заточена на то, чтобы добавлять, менять или удалять колонки в таблице.

Результат:

CatID	CatName	CatAge	CatColor	City
1	Мурка	3	Чёрная	Москва
3	Симба	5	Пурпурный	Санкт-Петербург
4	Лео	2	Полосатый	Екатеринбург
5	Мася	1	Серый	Москва

DROP TABLE

Если таблица больше не нужна, можем удалить её. Сделать это просто:

```
DROP TABLE CatsAndOwners;
```

Агрегатные функции

Агрегатные функции используют для того, чтобы производить вычисления с данными в таблице: считать количество строк, суммировать значения в столбце, найти среднее значение и так далее.

В SQL доступны пять агрегатных функций:

- COUNT — посчитать количество строк;
- SUM — посчитать сумму значений в столбце;
- AVG — получить среднее значение в столбце;
- MIN — получить минимальное значение в столбце;
- MAX — получить максимальное значение в столбце.

Вычислим совокупный возраст всех кошек:

```
SELECT SUM(CatAge) AS TotalAge
FROM CatsAndOwners;
```

Результат:

TotalAgeCatName
11

Теперь найдём наименьший возраст кошки:
SELECT MIN(CatAge) AS MinAge
FROM CatsAndOwners;

Результат:

MinAge
1

А теперь вычислим средний возраст кошек для каждого города:

SELECT City, AVG(CatAge) AS AverageAge
FROM CatsAndOwners
GROUP BY City;

Результат:

City	AverageAgeCatCount
Москва	2
Санкт-Петербург	5
Екатеринбург	2

Практическая часть.

Задание 1. Для БД "Каталог книг" составить следующие запросы (рис. 5.1):

1. Произведения, у которых нет авторов.
2. Авторы, у которых есть соавторы.
3. Произведения, у которых более одного автора.
4. Авторы, которые хотя бы одно произведение написали без соавторов.

5. Авторы, которые все произведения писали без соавторов.

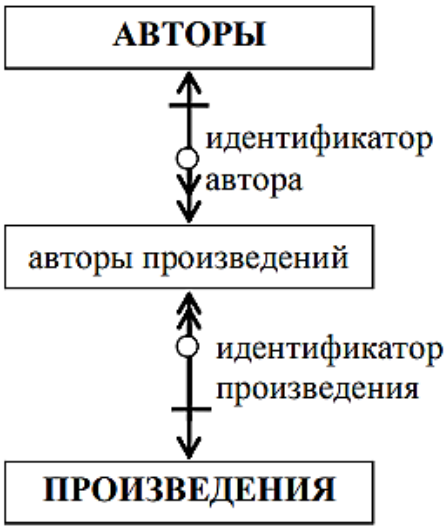

Рисунок 5.1. БД "Каталог книг"

Задание 2. Для БД "Больница" составить следующие запросы:

1. Список всех пациентов, которые в настоящее время лежат в больнице.
2. Список диагнозов пациентов, лечащими врачами которых являются хирурги.
3. Список всех врачей больницы с указанием отделения и специализации:
4. Список пациентов по отделениям.
5. Количество пациентов по палатам.
6. Вывести номера палат, в которых лежит только один пациент.
7. Проверить, что в одной палате не лежат и мужчины, и женщины.
8. Список пустых палат по отделениям.
9. Список палат, в которых нет свободных мест.
10. Список палат по отделениям, в которых есть свободные места, с указанием количества свободных мест.
11. Список палат отделения №2, куда может быть помещена вновь поступившая пациентка (женщина).

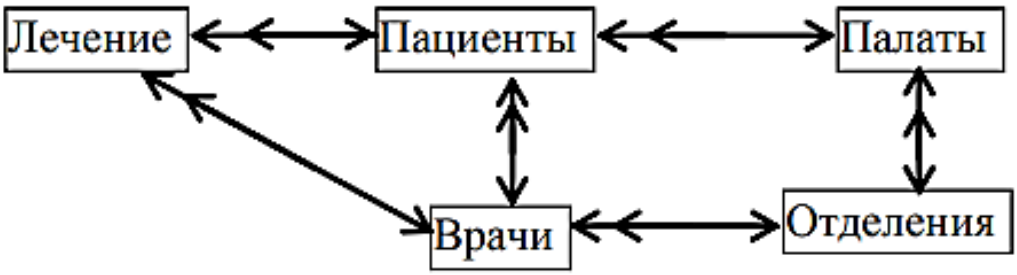

Рисунок 5.2. БД " Больница "

Контрольные вопросы:

1. Для каких целей используется язык SQL?
2. На какие группы делятся запросы языка SQL?
3. Как выглядит структура SQL-запроса?
4. Приведите примеры SQL-запросов?
5. Перечислите основные агрегатные функции SQL.