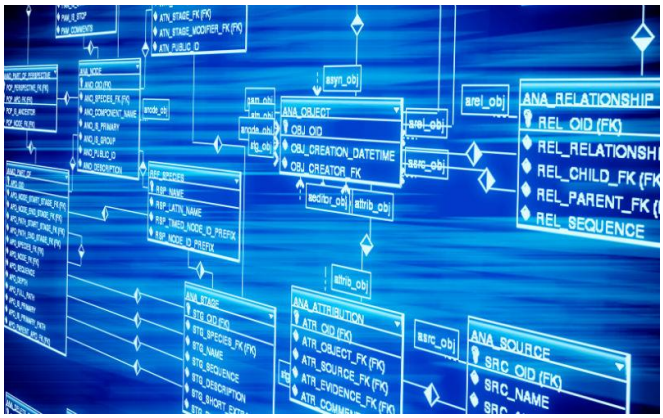
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

УПРАВЛЕНИЕ ДАННЫМИ

Учебно-методическое пособие
к выполнению лабораторных
и практических работ по дисциплине
для студентов, обучающихся по направлениям
09.03.02 «Информационные системы и технологии»
и 09.03.03 «Прикладная информатика»
по учебному плану подготовки бакалавров
очной и заочной форм обучения



Севастополь
СевГУ
2019

УДК 004.65 (075.8)
У67

Р е ц е н з е н т:
Е.Н. Мащенко - канд. техн. наук, доцент кафедры ИТиКС

Составители: Ю.В. Доронина, А.В. Волкова

У67 Управление данными: учеб.-метод. пособие к выполнению лабораторных и практических работ по дисциплине / Сост. Ю.В. Доронина, А.В. Волкова. – Севастополь: СевГУ, 2019. – 144 с.

Методические указания предназначены для проведения лабораторных и практических работ по дисциплине «Управление данными». Целью методических указаний является выработка у обучающихся практических навыков по работе с реляционными базами данных, помощь студентам в изучении процесса построения логической и физической моделей баз данных, возможностей и условий применения СУБД, в разработке интерфейса баз данных с использованием различного программного обеспечения и в получении навыков работы с инструментом MySQL Workbench, а также по настройке удаленного подключения к серверу баз данных. Излагаются теоретические и практические сведения необходимые для выполнения лабораторных работ, требования к содержанию отчета.

УДК 004.65 (075.8)

Методические указания рассмотрены и утверждены на заседании кафедры «Информационные системы» и методическом семинаре института информационных технологий и управления в технических системах, протокол № 7 от 28 января 2019 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В связи с широким внедрением вычислительной техники во все области деятельности человека, включая и повседневную жизнь, с одной стороны, и все увеличивающиеся потоки информации, с другой стороны, в настоящее время большую популярность получили различные электронные справочники, информационно - поисковые системы. В большинстве случаев, такие системы по своим размерам являются малыми и средними базами данных.

При разработке таких систем применяется теория реляционных баз данных, использующая сложный математический аппарат. Проектированию реляционных БД посвящено множество монографий, учебников и статей.

Данные методические указания предназначены для формирования навыков работы с распределенными реляционными БД. Предложенный комплекс лабораторных работ включает базовые принципы и подходы, которые необходимы для организации работы с современными средствами хранения информации. Так, в лабораторной работе №1 рассматриваются правила организации работы в среде многопользовательской реляционной БД Firebird, а также основные операторы языка описания данных и языка манипулирования данными.

Лабораторные работы № 2,4,5,6 - базовые операции по манипулированию данными, отображение реляционных операций на языке манипулирования данными SQL, простые и сложные (вложенные) запросы на выборку данных. Лабораторная работа №3 - требования к проектированию набора отношений реляционной БД для обеспечения целостности (достоверности) данных. Лабораторная работа №7 – средства для автоматического формирования значения ключа вводимых данных и способы создания автоматически вызываемых процедур, обеспечивающих дополнительные условия поддержания целостности данных при выполнении операций, изменяющих хранимые данные. Лабораторная работа №8 – манипулирование базой данных с помощью операций реляционной алгебры и языка SQL. Лабораторные работы №9 и 10 посвящены реализации всех этапов нормализации базы данных.

РАЗДЕЛ 1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ПО ПРОЕКТИРОВАНИЮ И МАНИПУЛИРОВАНИЮ БД

Общие определения и терминология

База данных (БД) – это совокупность взаимосвязанных, хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их оптимальное использование для одного или нескольких приложений. Данные организуются так, чтобы они были независимы от программных средств. Добавление (модификация) данных должна осуществляться общим управляемым способом.

Система управления базой данных (СУБД) – это программное обеспечение для использования и модификации БД одним или несколькими лицами.

Функции СУБД

1. **Главная**: обеспечить пользователя инструментарием, позволяющим оперировать данными в абстрактной форме, не связанным со способами хранения. СУБД представляет собой интерпретатор языка высокого уровня.
2. Обеспечение секретности.
3. Защита целостности данных (СУБД может осуществлять проверку на ограничение и не противоречивость данных).
4. Синхронизация (защита от одновременного доступа несколькими пользователями к одному элементу данных).
5. Защита от отказов и восстановление (обеспечение создания копий для восстановления БД после устранения причин сбоя).

Данные – это информация, фиксированная в определенной форме для последующей обработки, передачи и хранения.

Различают два аспекта данных:

1) инфологический;
2) даталогический.

Инфологический аспект связан со смысловым содержанием данных без учета их представления в памяти машины.

Даталогический аспект связан с представлением данных в памяти машины.

Объект – элемент, информация о котором сохраняется. Он может быть материальным и нематериальным.

Атрибут – свойство объекта.

Наименьшей единицей данных является отдельное значение данных, которое называется атомарным.

Домен – множество атомарных значений одного атрибута.

Отношение – таблица, обладающая следующими свойствами:

1) каждый элемент таблицы представляет собой один элемент данных (отсутствие повторяющихся групп);
2) все столбцы однородны (то есть элементы столбца имеют одинаковую природу);
3) столбцам однозначно присвоены имена;
4) в таблице нет двух одинаковых строк;
5) в операциях с такой таблицей все строку и столбцы могут рассматриваться в любом порядке не зависимо от их смысла и содержания.

Кортеж – строка такой таблицы.

Ключ (PRIMARY KEY, FOREIGN KEY, UNIQUE) – атрибут (совокупность атрибутов), используемый для идентификации записей или кортежа.

Основной ключ (PRIMARY KEY) – ключ, который используется для уникальной идентификации записей (первичный ключ).

Дополнительный (вторичный, уникальный) ключ (UNIQUE) – не идентифицирует уникальную запись, но идентифицирует все записи, обладающие одним определенным свойством.

Внешний ключ (FOREIGN KEY)– это атрибут или комбинация атрибутов отношения R2, значение которых обязательно должно совпадать со значением первичного ключа некоторого другого отношения R1.

Экземпляр записи – это значение в конкретной сроке таблицы.

Формальные термины	**неформальные термины**
отношения	таблицы
кортеж	запись, строка
атрибут	столбец

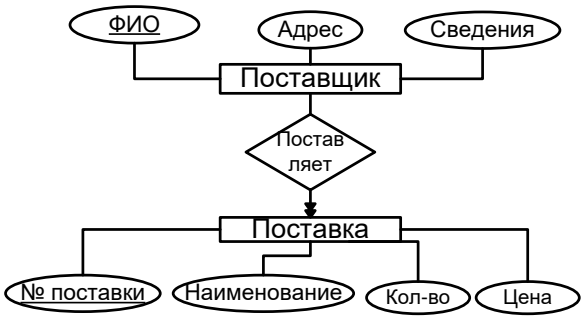
Рисунок 1 – Иллюстрация основных терминов БД

Схемой БД называется описание ее логической структуры . Схема представляет собой таблицу типов используемых данных, она содержит имена объектов и атрибутов и указывает связи между ними.

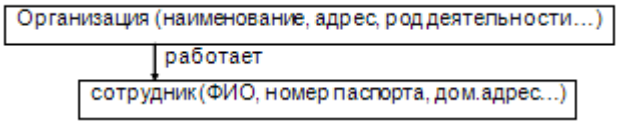
Схема состоит из блоков, в которых указываются поля записи, ключевое поле подчеркивается, над блоком ставится имя объекта. Блоки между собой связаны стрелками-связями.

Схемы БД могут изображаться в различных *нотациях* в зависимости от этапа проектирования БД:

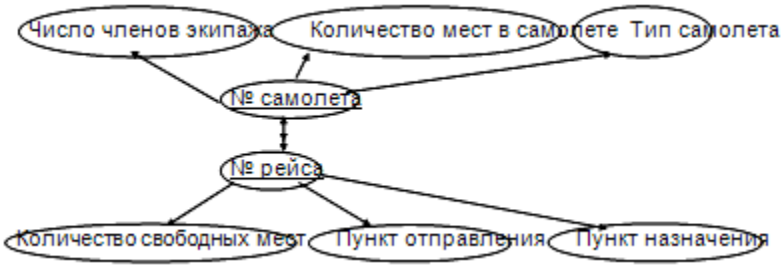
1. Этап описания данных (неформальные модели: ER - диаграммы; диаграммы Бахмана; овал – диаграммы).
2. Этап проектирования и нормализации (применяется нотаця П. Чена, IDEF1.X).



а)



б)



в)

Рисунок 2 – Неформальные модели:
а) ER – диаграммы (нотация П. Чена); б) диаграммы Бахмана; в) овал – диаграммы

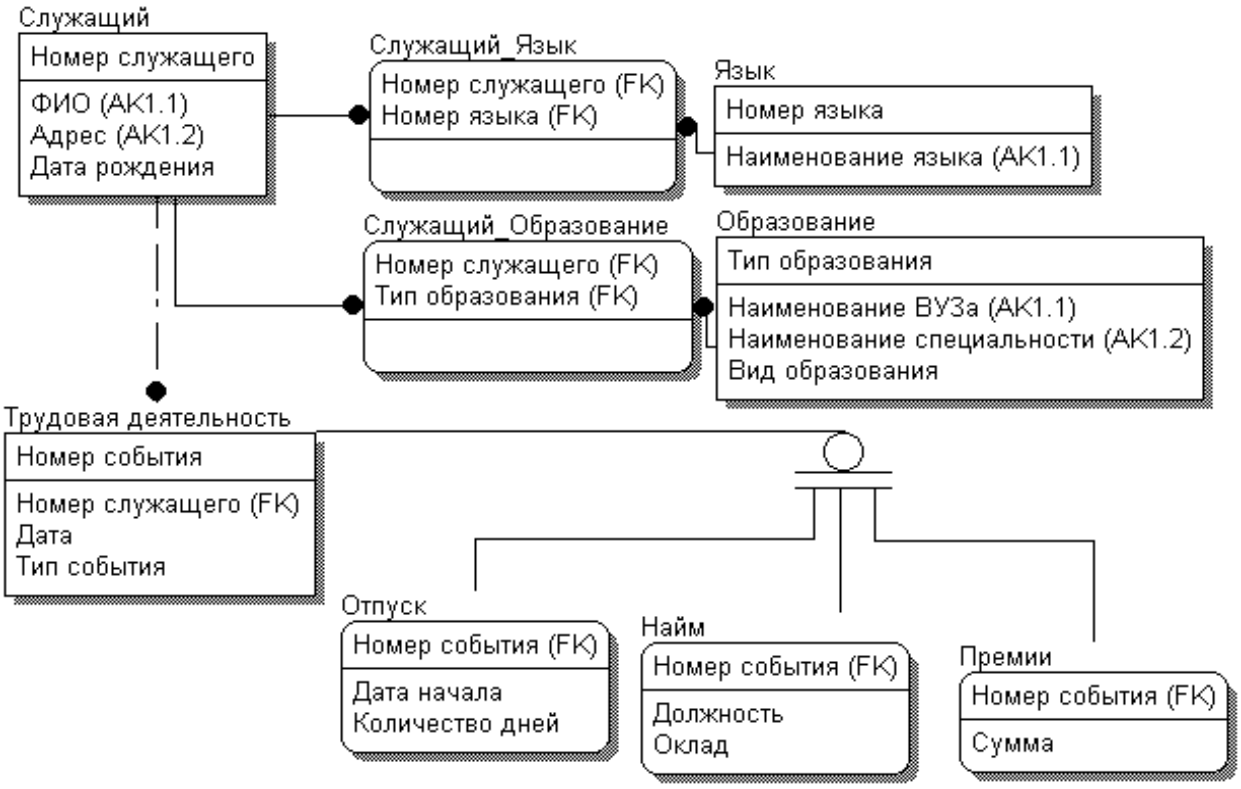


Рисунок 3 – Схема БД в нотации стандарта IDEF 1.X

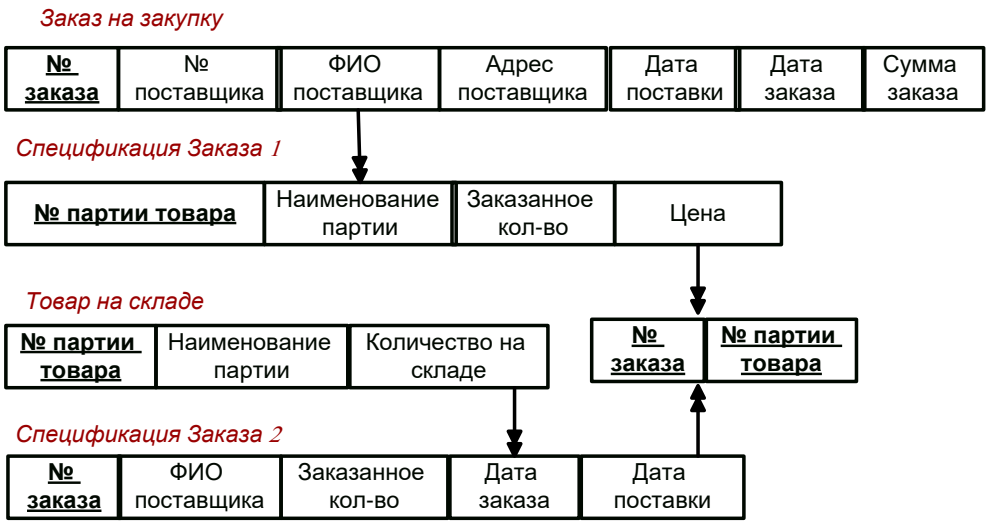


Рисунок 4 – Схема логической структуры БД для проектирования

Основой манипулирования данных являются специальные языки: ЯОД, ЯМД (DDL,DML).

ЯОД – язык высокого уровня, предназначенный для задания схемы БД, то есть описывающий типы данных, их структуру и взаимосвязь между ними. Это декларативный язык.

```
CREATE TABLE EMPLOYEE
      (EMP_NUM INTEGER NOT NULL,
       L_NAME CHAR(20) NOT NULL,
       N_NAME CHAR(10) NOT NULL,
       DEP_NUM INTEGER NOT NULL,
       JOB_CODE CHAR(10),
       PHONE_EXT INTEGER CHECK (PHONE_EXT BETWEEN 222 AND 444),
                                SALARY  DECIMAL(6,2) DEFAULT 0);
```

Emp_Num	L_Name	F_Name	Dep_Num	Job_Cod	Phone Ext	Salary
111	Ivanov	Ivan	10	engineer	226	890,50
112	Petrov	Stepan	10	admin		1456,96
113	Sidorova	Irina	20	engineer	442	920,48

Рисунок 5 – Фрагмент применения ЯОД для описания таблицы БД

ЯМД – язык запросов – система команд, осуществляющая манипулирование данными.

В зависимости от способа реализации ЯМД СУБД классифицируют как: СУБД с включенным языком; СУБД с базовым языком.

```
SELECT snum, MAX (amt) FROM Orders GROUP BY snum;
```

Результат выполнения запроса:

snum	
1001	9891.88
1002	5160.45
1003	1713.23
1004	1900.10

В качестве включенного языка используются языки программирования высокого уровня. В СУБД с базовым языком разрабатывается собственный алгоритмический язык, который кроме операций манипулирования данными должен выполнять арифметические операции и операции ввода/вывода.

Вопросы и задания по разделу 1

1. Дать определение базы данных (БД).
2. Дать определение системы управления базой данных (СУБД).
3. Назвать главную функцию СУБД.
4. Дать определение отношения БД.
5. Какие аспекты данных существуют?
6. Почему на этапе проектирования БД важен инфологический аспект данных?
7. На каком этапе создания БД учитывается даталогоческий аспект?
8. Что такое домен, кортеж БД?
9. В отношении БД в среднем больше кортежей или доменов?
10. Перечислить виды ключевых атрибутов.
11. Обосновать функцию ключевых атрибутов.
12. Какой атрибут в таблице R1 может быть выбран в качестве ключевого?

R1

N паспорта	ФИО	Отдел	Стаж
1111	Иванов И.И.	Технич.	10
2222	Иванова М.М.	Кадры	5
3333	Петров В.В.	Технич.	3

13. Какой атрибут может быть выбран в качестве внешнего ключа для отношений R1 и R2?

R2

Отдел	Начальник	Средний оклад
Технич.	Иванов И.И.	3000
Кадры	Сокол О.О.	2500
Проекты	Новиков А.А.	3500

14. Для отношений R1 и R2 предложить атрибут, который может быть дополнительным (вторичным) ключом.
15. Сколько доменов в отношении R1?
16. Сколько неповторяющихся элементов домена «Отдел» в отношениях R1 и R2 может быть? От чего это зависит?
17. Какие проблемы при описании доменов «Стаж» и «Средний оклад» вы можете сформулировать?
18. Перечислить нотации описания моделей и схем данных при создании БД.
19. Что такое ЯМД и ЯОД?
20. Как вы считаете, какая задача проектировщика БД связана с интерфейсом?

РАЗДЕЛ 2. ПРОЦЕСС ПРОЕКТИРОВАНИЯ БД

Цель проектирования: создать точную и защищенную БД, на основе которой можно гарантировать эффективное построение прикладных программ.

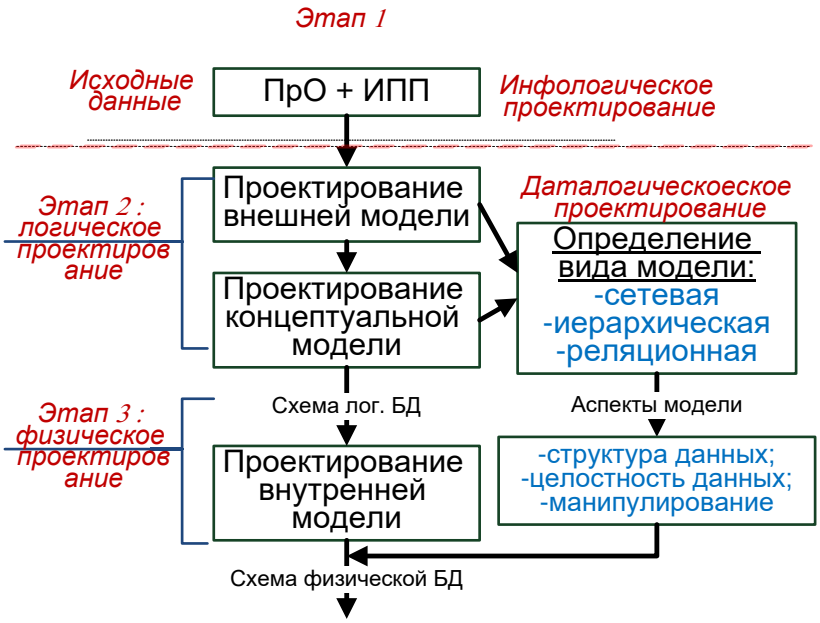


Рисунок 6 – Процесс проектирования БД

Рисунок 7 – Соотношение ПрО и ИПП
ИПП – информационные потребности пользователя; ПрО – предметная область

Процесс проектирования БД состоит из 2-х основных этапов:

1) проектирование логической БД;
2) проектирование физической БД.

При проектировании логической БД производится анализ ПрО и ИПП. Пользовательские требования выражаются рядом внешних моделей – представлений. Проектирование внешней модели заключается в формализации этих представлений.

Концептуальная модель данных (КМД) соответствует общему представлению о БД, то есть она включает представление о структуре данных, их целостности и манипулировании данными. Преобразование ***внешней модели (ВМД)*** в КМД определяется выбором СУБД. Как внешняя, так и концептуальная модель может быть 3-х видов:

1) сетевая;
2) иерархическая;
3) реляционная.

Физическое проектирование связано с фактической реализацией БД. Оно определяет рациональный выбор структуры хранения данных и методов доступа к ним. Результат физического проектирования - ***внутренняя модель данных***.

Реляционная БД – это набор экземпляров конечных отношений. Схема реляционной БД может быть представлена в виде совокупности следующих отношений:

$$\begin{cases} R1(A11, A12, ..., A1n) \\ ... \\ Rm(Am1, Am2, ..., Amk), \end{cases}$$

где *m, n, k* – любые целые числа, определяющие размерность схемы БД.

Главная задача проектирования реляционной БД: понизить избыточность и повысить надежность БД. При этом основное место отводится целостности данных, то есть ограничениям на зависимости между атрибутами отношений.

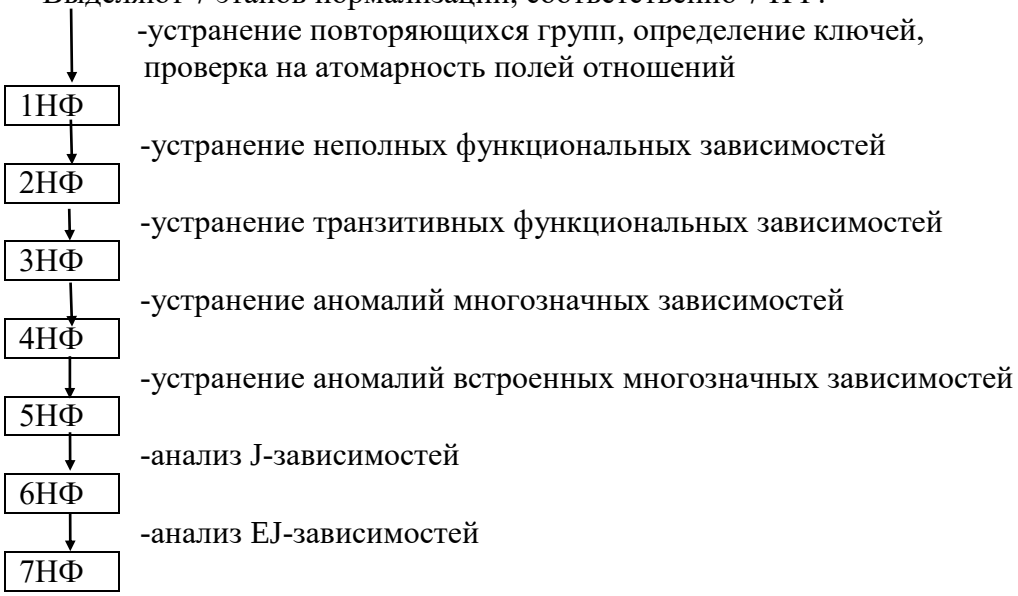
Нормализация необходима для решения проблемы рационального выбора вариантов схем отношений из возможного множества альтернативных вариантов.

Схема отношений должна удовлетворять следующим требованиям:

1) выбранные для отношения первичные ключи должны быть минимальными;
2) выбранный состав отношений БД должен быть минимален то есть отличаться минимальной избыточностью атрибутов;
3) при выполнении операций модификации, удаления и включения не должно быть трудностей;
4) перестройка набора отношений, приведение новых типов данных должна быть минимальной;
5) разброс времени ответа на различные запросы к БД должен быть небольшим.

Методы нормализации базируются на понятиях функциональных зависимостей, многозначных зависимостей и зависимостей соединения.

Выделяют 7 этапов нормализации, соответственно 7 НФ:



Окончательная цель нормализации – это получение такого проекта БД в которой каждый факт появляется лишь в одном месте, то есть max исключена избыточность информации (избыточность больше всего влияет не на объем памяти, а на удалении противоречивости данных).

Нормализованная таблица автоматически считается приведенной в 1НФ. Таким образом, понятие «нормализованная» и находящаяся в 1НФ эквивалентны.

Определение 1НФ: чтобы отношение соответствовало 1НФ необходимо привести это отношение в соответствие трем условиям:

1) определить первичные ключи;

2) устранить повторяющиеся группы;

3) привести поля отношения к атомарности. Основой этого этапа нормализации является определение ключей.

На рисунке 8 приведена схема определения ключевых атрибутов в древовидной форме, которая называется *правилом распространения ключей.*

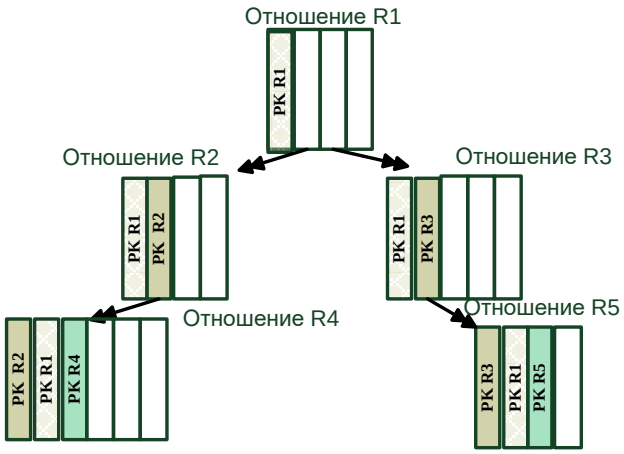


Рисунок 8 – Схема распространения ключей в древовидной структуре:
PK Ri – первичный ключ отношения i

Таблица находится во **2НФ**, если она удовлетворяет определению первой и все ее поля не входящие в первичный ключ связаны полной функциональной зависимостью с первичным ключом.

Рейс (№ самолета, дата вылета, время вылета, пункт назначения, тип самолета, грузоподъемность, число членов экипажа, обслуживаемая фирма, ФИО командира экипажа, звание командира экипажа)

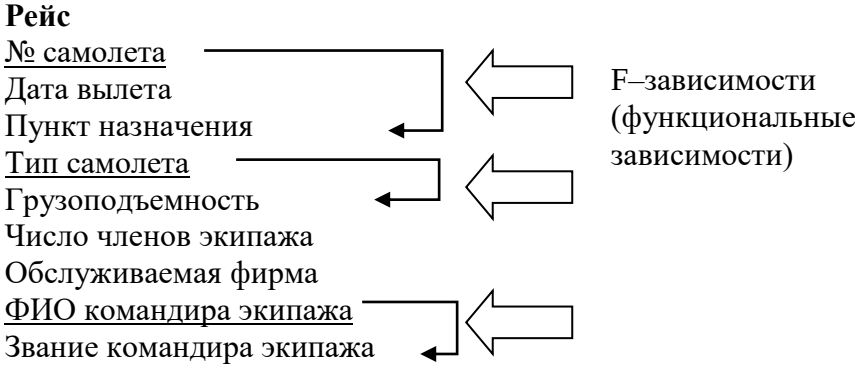


Таблица находится в **3НФ**, если она удовлетворяет **2НФ** и ни одно из ее не ключевых полей не зависит функционально от другого не ключевого поля, т.е. существуют зависимости между неключевыми атрибутами:

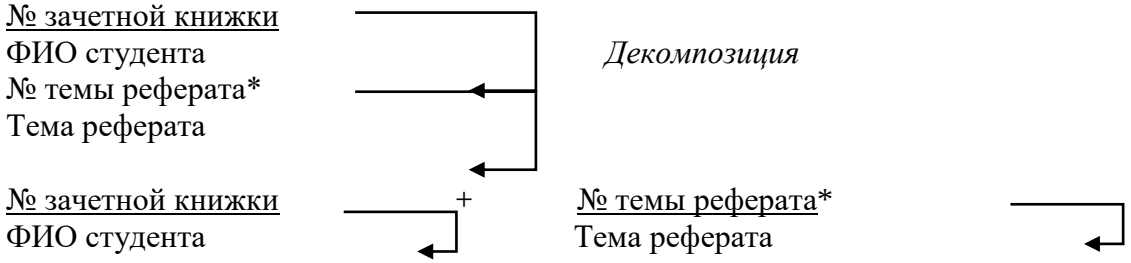


Таблица находится в **НФ Бойса-Кодда (НФБК)** тогда и только тогда, когда любая функциональная зависимость между полями сводится к полной функциональной зависимости от возможного ключа.

В соответствии с этим таблицы Блюда и Продукты, имеющие по паре возможных ключей (№ блюда, блюдо, № продукта, продукт) находятся в **НФБК** или в **3НФ**.

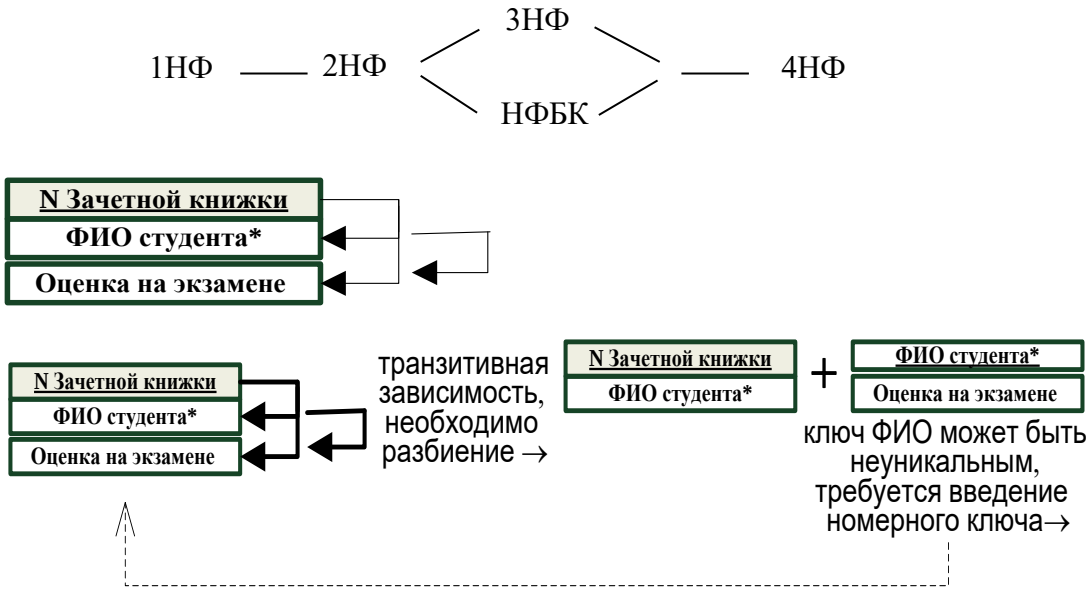


Рисунок 9 – Доказательство отсутствия декомпозиции в НФБК

В следующих **НФ** (4 и 5) учитываются не только функциональные многозначные связи между полями таблицы.

Полная декомпозиция таблицы – это совокупность любого числа ее проекций, соединение которых полностью совпадает с содержимым таблицы.

Обычно на практике достаточно приведения БД к 3НФ или НФБК.

Вопросы и задания по разделу 2

1. Назовите этапы проектирования БД.
2. Сформулировать цель проектирования БД.
3. Что такое предметная область и информационные потребности пользователя?
4. В чем отличия логической и физической моделей БД?
5. Что такое концептуальная и внешняя модели данных?
6. С чем связано физическое проектирование БД?
7. Записать формальное представление БД в виде совокупности отношений.
8. Дать определение реляционной БД.
9. Сформулировать главную задачу проектирования БД.
10. Сформулировать требования к отношениям БД.
11. Что такое нормализация БД?
12. Сколько существует основных нормальных форм?
13. Сформулировать основные этапы приведения к НФ.
14. Какова цель «распространения» ключей при приведении к 1НФ?
15. Почему при приведении к НФБК декомпозиция отношений не требуется?

Вопросы и задачи для самостоятельной работы

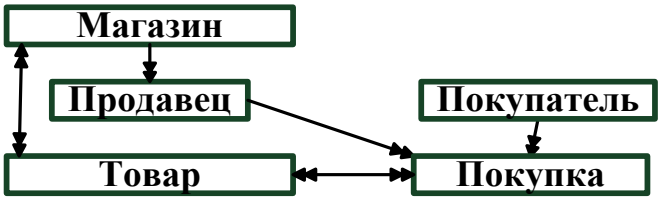
1. Сформулировать виды неатомарности полей в схемах и процесс приведения полей к атомарному типу.
2. Привести к атомарности:

ID	Наименование устройства	Дата поставки	Состав
012	Рабочая станция	01.12.12	Системный блок
			Монитор
			Клавиатура
			Принтер

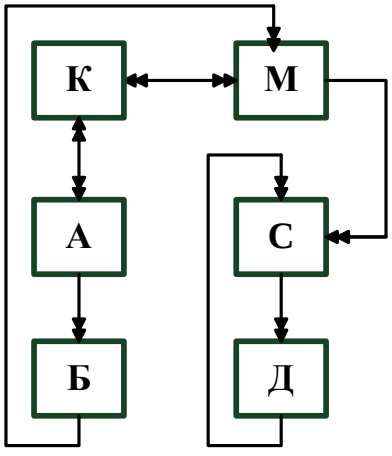
3. Привести к атомарности:

Адрес					Номер паспорта
Город	Улица	Дом	Корпус	Квартира	

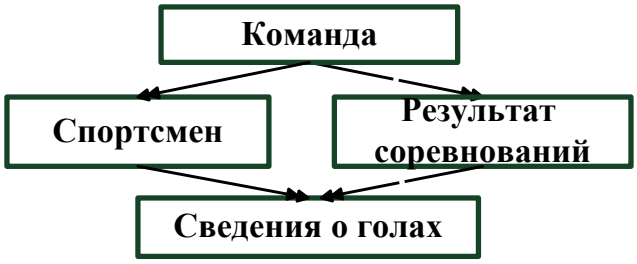
4. Что такое функциональная засисимость?
5. Какие типы зависимостей существуют, кроме функциональной?
6. Что такое ограничения целостности БД?
7. Определить 2НФ на основе формулировки о полноте F- зависимости.
8. Определить транзитивность на основании аксиом Армстронга.
9. Сформулировать 5НФ, 6НФ, 7НФ и ДКНФ.
10. Описать процесс приведения к древовидной структуре из сетевой.
11. Определить отличия сложной и простой сетевой структур.
12. Что такое «развязка» отношения М:М?
13. Преобразовать сложную сетевую структуру к древовидной:



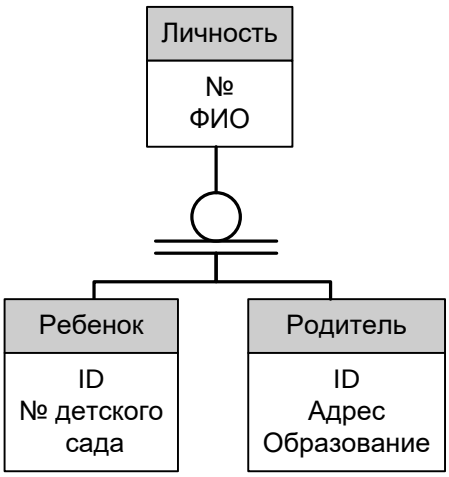
14. Показать процесс поэтапного преобразования сложной сетевой структуры, представленной в общем виде к древовидной:



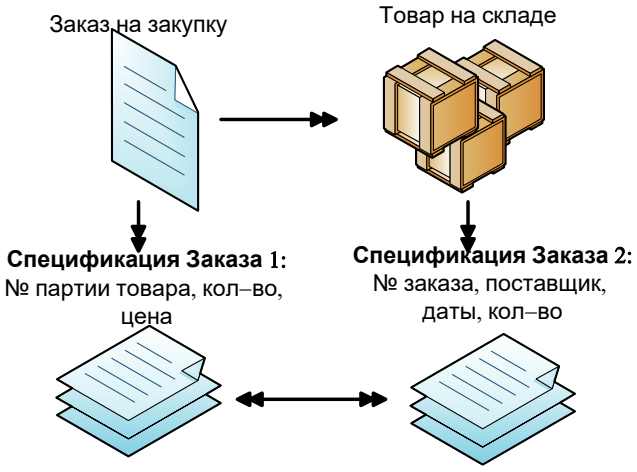
15. Определить возможные ключи отношения «сведения о голах»:



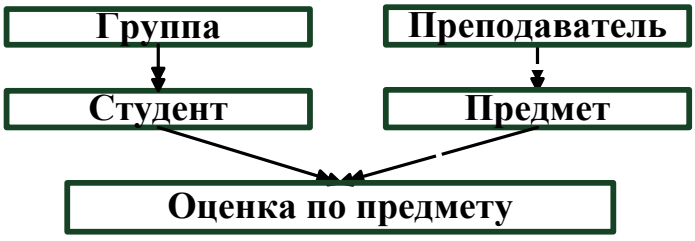
16. Как называется связь, изображенная на рисунке:



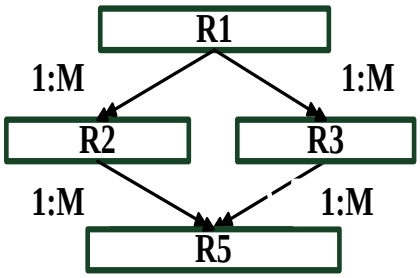
17. Какое количество отношений будет при нормализации приведенной ниже схемы?



18. Нормализовать схему отношений БД:



19*. Дана схема отношений БД в общем виде.



$R1(\underline{PK_1^1}, NK_1^1, NK_2^1, NK_3^1)$

$R2(\underline{PK_1^2}, NK_1^2, NK_2^2)$

$R3(\underline{PK_1^3}, NK_1^3, NK_2^3)$

$R5(\underline{PK_1^5}, PK_2^5, NK_1^5, NK_2^5)$

Нормализовать при условиях:

1) $F_{R5}: \{\underline{PK_1^3} \to NK_1^5; (\underline{PK}^5) \to NK_1^5, NK_2^5\}$
2) $F_{R2}: \{\underline{PK_1^2} \to NK_1^2; NK_1^2 \to NK_2^2\}$
3) $F_{R3}: \{\underline{PK_1^1} \to\to \underline{PK_1^3}; (\underline{PK}^3) \to NK_1^3, NK_2^3\}$
4) $R_1 \leftarrow\leftarrow \to\to R_2$

РАЗДЕЛ 3. ПРИМЕР НОРМАЛИЗАЦИИ ОТНОШЕНИЙ

В качестве примера реализации процесса нормализации, рассмотрим простейшее бизнес-приложение. В данном примере будет рассматриваться работа с упрощенной базой данных компании по разработке ПО. У каждого проекта имеются свой уникальный номер, название, штат сотрудников и т.д. У каждого сотрудника есть идентификационный номер и специальность, например, инженер-программист или постановщик задач.

Для учёта трудозатрат на разработку ПО компания периодически составляет отчеты, представляющие собой расходы на заработную плату сотрудников по каждому из проектов. Для этого учитываются человеко-часы, затраченные каждым работником на выполнение данного контракта, которые умножаются на стоимость одного часа работы соответствующего специалиста.

Рассмотрим сущность, содержимое которой соответствовало бы форме самого отчета (**Ошибка! Источник ссылки не найден.**). Причём, общий итог в **Ошибка! Источник ссылки не найден.** – это производный атрибут, который получается умножением отработанных часов на стоимость часа работы, и поэтому он не хранится в БД.

Таблица 1 – Периодически составляемый отчёт

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAM	JOB_CLASS	CHG_HOUR	HOURS	
Номер проекта	**Наименование проекта**	**Номер работника**	**Ф.И.О. работника**	**Должность**	**руб/ час**	**трудозатраты в часах**	**Общие расходы**
15	Alpha Edit	101	Семен Иванов	Программист	200	120	24000
		102	Андрей Петров	Программист	200	100	20000
		110	Антон Сидоров*	Сист. аналитик	300	40	12000
18	Beta Base	103	Федот Антонов	Программист	200	250	50000
		102	Андрей Петров	Программист	200	280	56000
		111	Петр Семенов*	Проектировщик БД	250	80	20000
22	Delta CAD	104	Сидор Федотов	Программист	200	180	36000
		105	Иван Андреев	Программист	200	150	30000
		110	Антон Сидоров*	Системный аналитик	300	60	18000
	Итого:						266000

Структура **Ошибка! Источник ссылки не найден.** не соответствует требованиям к реляционным базам данных, по причинам, представленным ниже.

1. Номер проекта (PROJ_NUM) нельзя использовать в качестве первичного ключа (или части первичного ключа) поскольку в нем имеются пустые места (null).

2. Вид элементов отношения стимулирует противоречивость данных. Например, значение JOB_CLASS (специализация) в одном случае введена как «Системный аналитик», в другом «Сист. аналитик».

3. Всущности имеется очевидная избыточность данных, что приводит к следующим аномалиям:

- *аномалии обновления:* изменение JOB_CLASS для сотрудника с номером (EMP_NUM) 102 требует выполнения этого действия в нескольких строках сущности, где встречается EMR_NUM = 102;
- *аномалии включения:* для занесения информации в любую строку необходимо вводить в проект какого-нибудь сотрудника. Если сотрудник еще не включен в какой-либо проект, то для того, чтобы занести в сущность сведения о сотруднике, придется создавать фиктивный проект;
- *аномалии удаления:* если сотрудник с номером 102 увольняется, то необходимо будет удалить все строки, тле EMP_NUM = 102. При этом может быть утеряна и важная информация.

Может показаться, что структура БД работает правильно, и выполнить отчет очень просто. Однако могут возникнуть некоторые проблемы.

Каждый раз, когда на проект назначается новый сотрудник, некоторые элементы данных (PROJ_NAME, EMP_NAME, CHG_HOUR) приходится вводить без всякой на то необходимости. Представьте себе объем ненужного ввода данных, если потребуется ввести 200-300 элементов сущности. Ведь достаточно ввести соответствующий идентификационный номер сотрудника, чтобы можно было идентифицировать, например, Антона Сидорова, его специализацию и стоимость часа его работы. Поскольку лишь один сотрудник имеет номер 110, его персональные данные (имя, специальность и т.д.) не должны набираться всякий раз при обновлении базы данных. Структура, представленная в **Ошибка! Источник ссылки не найден.**, не предоставляет такой возможности.

Избыточность данных, имеющая место в **Ошибка! Источник ссылки не найден.**, может привести к не экономному использованию дискового пространства. К тому же избыточность данных является причиной аномалий. При вводе новых данных, например, может возникнуть такая строка:

15	Alfa Edit	**111**	П.Семёнов	Проектировщик БД	**280**	**0**

На первый взгляд эти данные корректны. Но разве, Alfa Edit это тот же проект что и Alpha Edit? А П. Семёнов – та же личность, что и Петр Семенов? Подобная путаница вызвана проблемой целостности данных, и произошла она потому, что при вводе информации не соблюдалось правило, предписывающее, что все копии избыточных данных должны быть идентичны.

При проектировании БД проблемы целостности данных, которые могут быть вызваны их избыточностью, обязательно должны приниматься во внимание.

Приведение к первой нормальной форме. В реляционной БД не должно быть повторяющихся групп. Если же такие группы существуют, то от них необходимо избавиться для того, чтобы каждая строка определяла единственное значение сущности. Процедура заключается в добавлении соответствующего компонента, по крайней мере, в столбец (или столбцы) первичного ключа. Это означает, что данные, представленные в **Ошибка! Источник ссылки не найден.**, должны быть преобразованы к виду, представленному в **Ошибка! Источник ссылки не найден.** Удалив повторяющиеся группы, мы получаем БД, приведенную к первой нормальной форме.

По изменениям, представленным в **Ошибка! Источник ссылки не найден.** видно, что атрибут PROJ_NUM нельзя использовать в качестве первичного ключа, поскольку номер проекта не идентифицирует уникально все остальные атрибуты сущности (строки). Например, значение PROJ_NUM, равное 15, может определять одного из пяти сотрудников. Первичный ключ, уникально определяющий любое значение атрибута, должен быть комбинацией атрибутов PROJ_NUM и EMP_NUM.

Таблица 2 – Организация данных: первая нормальная форма (1НФ)

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Alpha Edit	101	Семен Иванов	Программист	200	120
15	Alpha Edit	102	Андрей Петров	Программист	200	100
15	Alpha Edit	110	Антон Сидоров*	Сист. аналитик	300	40
18	Betta Prog	103	Федот Антонов	Прграммист	200	250
18	Beta Prog	102	Андрей Петров	Программист	200	280
18	Beta Prog	111	Петр Семенов*	Проектировщик БД	250	80
22	Delta CAD	104	Сидор Федотов	Программист	200	180
22	Delta CAD	105	Иван Андреев	Программист	200	150
22	Delta CAD	110	Антон Сидоров*	Системный аналитик	300	60

Зависимости можно установить с помощью *диаграммы зависимостей,* представленной на Рисунок 10.

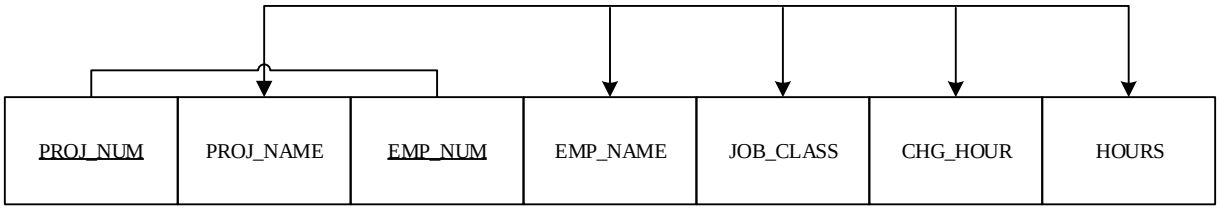


Рисунок 10 – Диаграмма зависимостей, основанных на первичном ключе

На Рисунок 10 необходимо обратить внимание на следующее:

1) атрибуты первичного ключа подчеркнуты;

2) линии со стрелками над сущностями указывают все возможные желательные зависимости, т.е. зависимости, основанные на первичном ключе.

В БД имеется составной ключ, в который входят атрибуты PROJ_NUM и EMP_NUM. Любой атрибут, который, по меньшей мере, является частью ключа, называется *первичным атрибутом* или *ключевым атрибутом.* Следовательно, и PROJ_NUM, и EMP_NUM являются первичными (ключевыми) атрибутами. Соответственно, непервичный атрибут (или неключевой атрибут) не является частью ключа.

Говорят, что БД приведена к первой нормальной форме (1НФ), если в ней:

- определены все ключевые атрибуты;
- отсутствуют повторяющиеся группы. Иначе говоря, на пересечении каждого столбца и каждой строки содержится только одно значение (см. свойства реляционных таблиц), а не множество значений;
- все атрибуты зависят от первичного ключа.

Таким образом, сущность, представленная в **Ошибка! Источник ссылки не найден.** удовлетворяет требованиям, предъявляемым к 1НФ.

Приведение ко второй нормальной форме. В сущности, представленной на Рисунок 10 имеются необязательные зависимости. Эти зависимости называются *частичными* и указаны линиями со стрелками в нижней части диаграммы зависимостей, представленной на Рисунок .

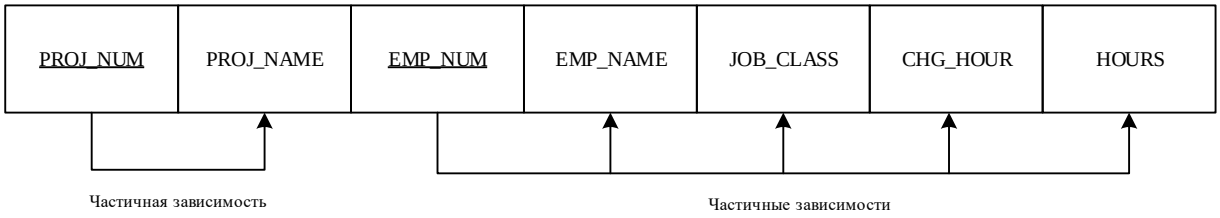


Рисунок 11 – Диаграмма частичных зависимостей

Необходимо знать только значение PROJ_NUM для определения PROJ_NAME, т.е. PROJ_NAME зависит только от части первичною ключа. А для определения EMP_NAME, JQB_CLASS и CHG_HOUR необходимо знать только EMP_NUM. Зависимость, определяемая только частью составного первичного ключа, называется частичной зависимостью.

Для преобразования 1НФ в 2НФ необходимо взять за основу форму 1НФ, представленную на Рисунок и проделать следующее:

- записать каждый ключевой компонент и отдельной строке, а затем в последней строке записать исходный (составной) ключ:

PROJ_NUM
EMP_NUM
PROJ_NUM EMP_NUM

− каждый компонент станет ключом в новой сущности. Иначе говоря, исходную сущность необходимо разделить на три сущности. Назовем эти таблицы PROJECT (проект), EMPLOYEE (сотрудник) и ASSIGN (назначение) соответственно;

− после каждого нового ключа записать зависимые атрибуты. Зависимости для компонентов исходного ключа можно выявить, прослеживая линии со стрелками в нижней части диаграммы зависимостей. Другими словами, три новые сущности PROJECT, EMPLOYEE и ASSIGN могут быть описаны следующим образом:

PROJECT (**PROJ_NUM**, PROJ_NAME)

EMPLOYEE (**EMP_NUM**, EMP_NAME, JOB_CLASS, CHG_HOUR)

ASSIGN (**PROJ_NUM**, **EMP_NUM**, ASSIGN_HOURS)

− поскольку количество часов, затраченных на каждый проект каждым сотрудником, зависит в сущности ASSIGN как от атрибута PROJ_NUM, так и от атрибута EMP_NUM, мы поместим их в сущность ASSIGN под названием ASSIGN_HOURS.

Результат этой операции представлен на Рисунок 12. Теперь большинство аномалий, которые мы ранее обсуждали, устранено. Например, если мы захотим добавить/изменить/удалить запись в проекте PROJECT, то необходимо лишь обратиться к сущности PROJECT и добавить/изменить/удалить только одну строку.

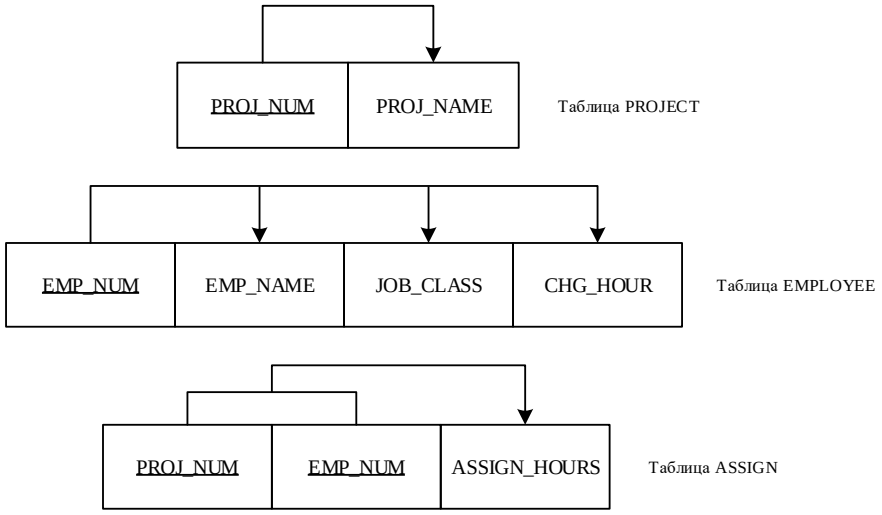


Рисунок 12 – Результат приведения ко второй нормальной форме (2НФ)

Говорят, что таблица приведена ко второй нормальной форме (2НФ), если:

− она приведена к первой нормальной форме;

− в ней нет частичных зависимостей, т.е. нет атрибутов, зависящих от части первичного ключа.

Приведение к третьей нормальной форме. На Рисунок сущности EMPLOYEE видна транзитивная зависимость (см. Рисунок), которая может стать причиной аномалий. Например, если стоимость часа работы для данной специализации, которую имеют несколько сотрудников, изменится, то это изменение необходимо будет проделать для *каждого* такого сотрудника. Если вы забудете обновить какие-то записи, связанные с изменением стоимости часа, то разные сотрудники, имеющие одну и ту же специализацию, получат разную зарплату.

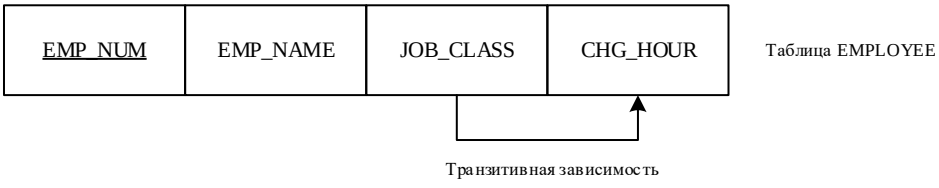


Рисунок 13 – Диаграмма транзитивных зависимостей

На Рисунок 13 видно, что CHG_HOUR зависит от JOB_CLASS. Поскольку ни CHG_HOUR, ни JOB_CLASS не являются первичными атрибутами – т е. ни один из этих атрибутов не является, по крайней мере, частью ключа – в этом случае говорят, что имеет место транзитивная зависимость. Проблема здесь состоит в том, что такие зависимости тоже могут стать причиной аномалии данных.

Аномалии данных, созданные в базе данных, представленной на Рисунок , можно устранить, разбив фрагменты или фрагмент, на который указывает линия со стрелкой транзитивной зависимости (в нижней части диаграммы зависимостей) и поместив их в отдельную сущность. Однако атрибут JOB_CLASS должен остаться в исходной сущности (имеющей 2НФ) в качестве внешнего ключа для того чтобы, установить связь между исходной сущностью и вновь созданной. Иначе говоря, после завершения преобразования в базе данных должно быть четыре сущности (см. Рисунок):

PROJECT (PROJ_NUM, PROJ_NAME)
ASSIGN (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)
EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS)
JOB (JOB_CLASS, CHG_HOUR)

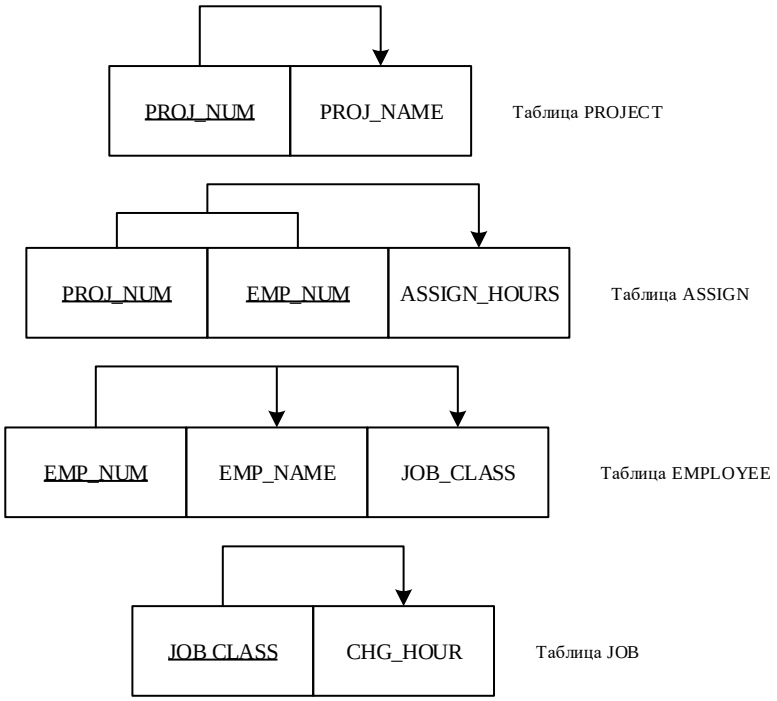


Рисунок 14 – Результат приведения к третьей нормальной форме (3НФ)

Выполнив это преобразование, мы устранили транзитивную зависимость оригинальной таблицы EMPLOYEE, и теперь таблица приведена к третьей нормальной форме (3НФ).
Говорят, что сущность приведена к *третьей нормальной форме (3НФ),* если:

- она приведена ко второй нормальной форме (2НФ);
- в ней отсутствуют транзитивные зависимости.

Таким образом, можно сделать вывод, что база данных находится в 3НФ.

РАЗДЕЛ 4. ЯЗЫКИ МАНИПУЛИРОВАНИЯ БД. ЯЗЫК SQL

Язык SQL является языком манипулирования для реляционной БД. В SQL реализовано три основных функции манипулирования данными:

1) определение;
2) выборка;
3) обновление.

Определение

```
CREATE TABLE <имя таблицы>(<определение столбца>
                          [,<определение столбца>...]);
<определение столбца>::=<имя столбца> <тип данных> [NOT  NULL];
<тип данных>::=INTEGER | SMALLINT | DECIMAL(p,q) | CHAR(n) | VARCHAR(n) |
FLOAT
```

Выборка

```
SELECT [DISTINCT] <элементы> FROM<таблица(ы)> WHERE <предикат>
[GROUP BY <поле(ля)> [HAVING <предикат>]]
[ORDER BY <поле 1(ля1)>];
< элемент >::=* | <имя  таблицы>.* | <арифметическое выражение>|
<стандартная функция>| <имя столбца>| <имя таблицы>.<имя столбца>;
<стандартная функция>::= COUNT | SUM |AVG | MAX | MIN;
<таблица>::= <имя таблицы> |  <псевдоним>;
<поле 1>::= (<поле>)(ASC | DESC | по умолчанию ASC);
<поле>::= <номер столбца> |  <имя столбца>;
<предикат>::= <условие> |  <условие><знак предиката> | NOT <предикат>;
<знак предиката>::= OR | AND ;
<условие>::= <условие сравнения> | <условие between> |
<условие like> |  <условие in> |  <условие exists>;
<условие in>::= <имя столбца> [NOT]  IN (<числовые значения>) | <SELECT>;
```

Обновление

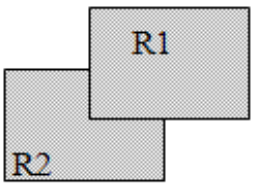
```
UPDATE <имя таблицы>
SET  <имя столбца> = <выражение>
[,<имя столбца> = <выражение>... ]  [WHERE <предикат>];
<выражение>::=<константа>|<арифметическое выражение> |  NULL
```

Манипулирование БД: реляционная алгебра (РА)

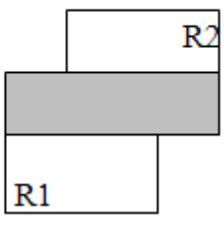
Две группы операций РА:

1. Традиционные теоретико-множественные операции применительно к отношениям:

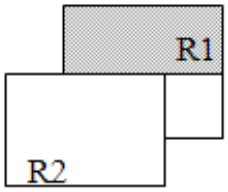
1) Объединение: $R=R1 \cup R2$



2) Пересечение: $R=R1 \cap R2$



3) Разность: R=R1-R2



4) Декартово произведение: R=R1×R2

a b c	×	x y	=	a x a y b x b y c x c y

2. Специальные реляционные операции:

5) Селекция: R = σ F (R1), где F - условие образования.

Формула условия F образуется:

а) операндами, являющимися именами (номерами) атрибутов (столбцов), для которых заданы ограничения области определения с помощью операций сравнения чисел:

б) логическими операциями «и», «или», «не»;

в) операциями сравнения чисел > , <, = , <=, >=.

6) Проекция: R = πi1...ik (R1), где i1...ik - атрибуты отношения R1 (номера или имена столбцов).

R1

a1	a2	a3	a4

R = π a1,a3 (R1)

Примечание:
Столбцы, которые войдут в отношение R, заштрихованы.

7) Соединение отношений : R=R1 ⊳⊲ R2, где

$$i\ \theta\ j$$

где i, j номера или имена столбцов в R1 и R2 соответственно,

θ - операция сравнения.

$$\mathbf{R = R1 \rhd\lhd R2 = \sigma_{i\ \theta\ (n+j)}\ (R1 \times R2),}$$

где n – арность отношения R1.

Если θ - операция равенства, то указанную операцию называют операцией **эквисоединения**. Ниже приведен пример такой операции.

R1

A	B	C
а	в	с
а	и	р
д	е	ж

R2

Д	E
а	и
е	к

R = R1 ⊳ ⊲ R2
B=D

A	B	C	Д	E
д	е	ж	е	к

Естественное соединение: **R = R1 ⊳⊲ R2**

R1 = (a1...ak , b1...bn)

R2 = (a1...ak , c1...cm)

R = R1 ⊳⊲ R2 = π a1...ak , b1...bn, c1...cm (σ (a1R1 = a1R2 ∧ ... ∧ akR1 = akR2)(R1 × R2))

a1	b1
a2	b1
a3	b2

+

b1	c1
b2	c2
b3	c2

=

a1	b1	c1
a2	b1	c1
a3	b2	c2

Естественное соединение

8) Деление: R = R1 : R2

n - арное m – арное

R1

a	x
a	y
a	z
b	x
c	y

R2

x
z

R

a

$R=\pi_{1,2\ldots(n-m)}(R1)-\pi_{1,2\ldots(n-m)}((\pi_{1,2\ldots(n-m)}(R1\times R2)-R1)$

Вопросы и задания по разделу 4

1. Найти отношение, обладающее свойством: $R=\sigma_{3=p}(R_1\cap R_2)$

R1

а	ж	с
а	и	р
д	е	ж

R2

с	ж	а
а	и	р
ж	е	д

2.Найти отношение и пояснить смысл операции: $R=\pi_A(R_1 \triangleright\triangleleft R_2)$ **В=Д**

R1

А	В	С	К
а	ж	с	д
а	и	р	у
д	е	ж	и

R2

Д	Е	Н
а	и	и
е	к	к
ж	у	в

3. Найти и определить операцию: R = R1 $\triangleright\triangleleft$ R2

R1

А	В	С	К
а	ж	с	д
а	и	р	у
д	е	ж	и

R2

А	М	Н	С
а	к	и	с
а	к	к	р
д	у	в	ж

4. Для варианта задания п. 3 найти: R=(R1 $\triangleright\triangleleft_{B=E}$ R2); R`=σA=д(R).

5. Найти отношение, обладающее свойством: $R=\pi_2 (R_1 \cap R_2)$

R1

а	ж	с
а	и	р
д	е	ж

R2

с	ж	а
а	и	р
ж	е	д

6. Найти: $R = \sigma_{(C=c)\wedge(A=a)}(R_1 \triangleright\triangleleft R_2)$

R1

А	В	С	К
а	ж	с	д
а	и	р	у
д	е	ж	и

R2

Д	Е	К
а	и	д
е	к	у
ж	у	и

7. Построить отношение по заданному свойству: $R=\sigma_{2=и} (R_1 * R_2)$

R1

а	ж	с
а	и	р
д	е	ж

R2

с	ж	а
а	и	р
ж	е	д

8. Для варианта задания п.7 найти отношения R3 и R4, обладающие свойствами:
1) $R3=R_1 - R_2$, 2) $R4=\pi_2(R1 \times R_2)$.

9. Для варианта задания п.7 найти отношения, обладающие свойствами:
1) $R3=R_1 \cap R_2$; 2) $R4=R_2-R_1$

10. Найти отношение: R=R1:R2.

R1

и	о	р	к
о	р	с	а
с	к	р	к
и	о	с	а
с	к	с	а
и	о	к	с

R2

с	а
р	к

Пояснить, каким образом получен результат.

11. Дана реляционная модель:

P1(N_аквариума, общее_количество_рыб, температура, тип_аэрации, тип_флоры);

P2(вид_рыбы, вид_корма, код_содержания, страна_происхождения);

P3(код_содержания, минимальная_температура, максимальная_температура, тип_флоры, тип_аэрации);

P4(N_аквариума, вид_рыбы, количество).

1) применить к БД реляционную операцию вида: π, \ (в одном запросе),

- сформулировать запрос;
- представить запись в реляционной форме;
- представить в форме запроса на SQL.

2) составить запросы (на РА и в форме SQL):

- получить список рыб для особей, чьей родиной является Бирма и температура содержания минимальна;
- сформулировать представление «теплолюбивые рыбы» на базе отношения P2;
- получить для каждого вида рыбы условия ее содержания и тип аэрации;
- удалить все записи из отношения P4;
- сформулировать и записать запрос на SQL, не реализующийся на РА.

3) представить ER-диаграмму данной реляционной модели.

Вопросы и задания для самостоятельной работы:

1. Изучить основы реляционного исчисления.
2. Эквивалентны ли данные выражения? Пояснить их смысл.

1) $R1=\sigma_{K=1}\ \pi_c\ (\pi_в\ (\sigma_{C=a}\ (R \triangleright\triangleleft S)))$

2) $Q=\{C,A|(\exists K),(\exists C),(\exists B)(R(a,b)\wedge S(b,c))\}$

3.Упростить выражение:

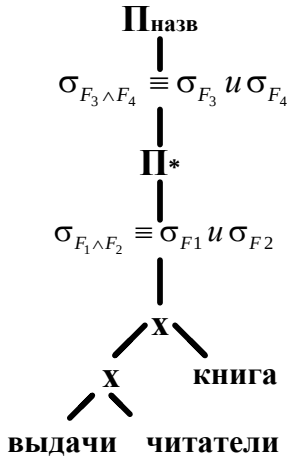
$Q=\{a_1,a_3,a_2|(\exists\ b_1),(\exists\ b_2),(\exists\ b_3),(\exists\ b_4)$

$(R(b_1,a_1,a_2)\wedge R(b_4,c_2)\wedge R(b_4,a_1)\wedge\ \ R(b_1,a_1)\wedge R(b_2,c_1)\wedge R(b_5,a_4,a_3,a_2))\}$

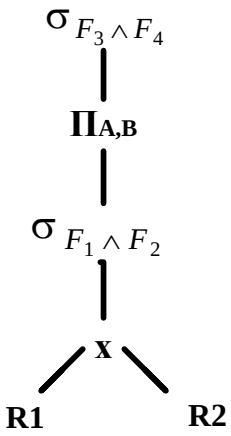
4. Изучить принципы оптимизации запросов.
5. При каких условиях истинно выражение:

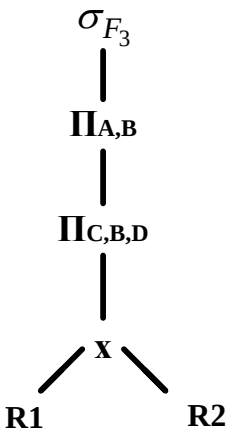
$\sigma_{F1}(\pi_{A1\ldots Am}(\sigma_{F2}(E))) \equiv \pi_{A1\ldots Am}\ (\sigma_F(\pi_{A1\ldots Am\ B1\ldots Bm}(E)))$?

6. Сформулировать основные виды эквивалентных преобразований.
7. Пояснить, какие операции из эквивалентных преобразований запросов могут быть применены к дереву запроса:



8. Упростить следующие деревья запросов:





ЛАБОРАТОРНАЯ РАБОТА № 1
ИЗУЧЕНИЕ ОСНОВ ЯЗЫКА МАНИПУЛИРОВАНИЯ ДАННЫМИ SQL НА БАЗЕ СЕРВЕРА FIREBIRD

1.1 Цель работы

Изучить основы организации сервера Firebird; научится устанавливать соединение с сервером. Научиться создать базу данных и производить элементарные действия над ней. Изучить формы оператора CREATE TABLE и простейшие формы оператора SELECT.

1.2 Основные положения

1.2.1 Описание лабораторной установки

Сервер Firebird является полнофункциональным сервером реляционных баз данных. Сервер выпускается под руководством несколько платформ (Windows, Linux, Solaris, freebsd, Darwin) в двух архитектурах – Super Server и Classic. Основное различие между ними состоит в том, что Classic создает параллельный процесс для каждого присоединяемого пользователя, а SuperServer состоит из одного процесса, который обрабатывает запросы клиентов в разных нитях (threads) этого же процесса. Архитектура Classic считается более надежной, а SuperServer более производительной.

Сервер на Windows запускается в качестве службы. Клиентские приложения могут присоединяться к нему несколькими способами: по протоколам NetBEUI, TCP/IP; локальное подключение (в случае, если вы работаете на машине, на которой запущен сервер). В дальнейшем рассматривается подключение к локальному серверу.

1.2.2 Создание и регистрация базы данных

Для создания базы данных необходимо запустить утилиту *isql*, которая находится в директории *bin* папки *Firebird* или через меню Пуск. Появится окно, представленное на рисунке 1.1.

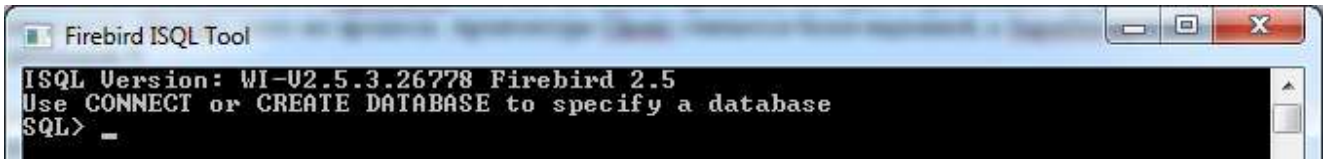


Рисунок 1.1 – Запуск утилиты *isql*

Оператор CREATE DATABASE создает новую базу данных:

```
SQL> create database 'c:\work\Ivanov.fdb'
CON> user 'SYSDBA' password 'masterkey'
CON> page_size = 4096
CON> default character set win1251;
```

Необходимо указать следующую информацию:

- имя файла (File Name) – имя физического файла базы данных, а также указать путь сохранения файла на диске;
- Имя пользователя (User Name) – SYSDBA (SYStem DataBase Administrator);
- пароль (Password) – masterkey;
- размер страницы базы данных задается равным 4096 (одно из допустимых значений).

Обязательно надо установить набор символов по умолчанию (Default character set) в UTF8 или WIN1251.

Для соединения с уже существующей базой данных используется оператор CONNECT:

```
SQL> connect 'c:\work\Petrov.fdb' user 'SYSDBA' password 'masterkey';
```

После создания БД или соединения с уже существующей можно приступать к работе с ней.

1.2.3 Описание базовых операторов для работы с таблицами

CREATE TABLE создает новую таблицу, ее столбцы и ограничения целостности в существующей базе данных. Пользователь, который создает таблицу, становится владельцем таблицы и получает полные привилегии на таблицу. Синтаксис оператора CREATE TABLE приведен в приложении Б. В общем виде описание оператора можно представит следующим образом:

```
CREATE TABLE <имя таблицы> (<опред_столбца> [,<опред_столбца> | <ограничения> ...]);
```

Обязательно надо указать имя таблицы и определить как минимум один столбец.

CREATE TABLE поддерживает следующие возможности для определения столбцов:

Столбцы определяют имя и тип данных для данных, вводимых в столбец. Основанные на доменах столбцы наследуют все характеристики домена. На столбец может определить значение по умолчанию, атрибут NOT NULL, дополнительные ограничения CHECK или порядок сортировки.

Вычисляемые столбцы. Значение столбца вычисляется каждый раз при доступе к таблице. Если тип данных не определен, Firebird определяет тип данных результата операции. Столбцы, к которым обращается вычисление, должны быть определены раньше, чем вычисляемый столбец.

Описание типа данных для столбца типа CHAR, VARCHAR или BLOB может включать предложение CHARACTER SET, определяя специфическую кодировку для одиночного столбца. Иначе **столбец** использует определенную по умолчанию для базы данных кодировку. Если кодировка базы данных изменена, все столбцы впоследствии определенные имеют новую кодировку, но существующие столбцы не изменяются.

Примеры использования оператора CREATE TABLE. Рассмотрим таблицу «Служащий» (EMPLOYEE), имеющую следующую структуру.

Emp_Num – номер служащего, целочисленное, не может принимать значение NULL;

L_Name – фамилия, строка переменной длины с максимальной длиной 20 символов, не может принимать значение NULL;

F_Name – имя, строка переменной длины с максимальной длиной 10 символов, не может принимать значение NULL;

Dep_Num – номер отдела, целочисленное, не может принимать значение NULL;

Job_Cod – должность, строка переменной длины с максимальной длиной 10 символов;

Phone Ext – внутренний номер телефона, целочисленное, находится в переделах от 222 до 444;

Salary – оклад, значение от 0 до 9999,99.

Таблица «Служащий»

Emp_Num	L_Name	F_Name	Dep_Num	Job_Cod	Phone Ext	Salary
111	Ivanov	Ivan	10	engineer	226	890,50
112	Petrov	Stepan	10	admin		1456,96
113	Sidorova	Irina	20	engineer	442	920,48

Следующая инструкция создает указанную таблицу:

```
CREATE TABLE EMPLOYEE
      (EMP_NUM INTEGER NOT NULL,
        L_NAME VARCHAR(20) NOT NULL,
        N_NAME VARCHAR(10) NOT NULL,
        DEP_NUM INTEGER NOT NULL,
        JOB_CODE VARCHAR(10),
        PHONE_EXT INTEGER CHECK(PHONE_EXT BETWEEN 222 AND 444),
        SALARY  DECIMAL(6,2) DEFAULT 0
   );
```

Таблица «Отдел» (DEPARTMENT) содержит следующие атрибуты:

Dep_Num – номер отдела, целочисленное, не может принимать значение NULL;

Dep_ Name – наименование отдела, строка переменной длины с максимальной длиной 20 символов;

Dep_ Head – номер служащего, руководителя отдела целочисленное;

Budget – бюджет, значение от 0 до 999999,99;

Location – адрес отдела, строка переменной длины с максимальной длиной 20 символов;

Phone_Num – номер телефона, целочисленное.

Таблица «Отдел»

Dep_Num	Dep_Name	Dep_Head	Budget	Location	Phone_Num
10	Inspector		20000,00	fl 5, of 16-34	454601
20	construct	164	500000,0	fl 6-7	456701
30	security	46	100000,0	fl 10, of 1-22	442101

Следующая инструкция создает указанную таблицу:

```
CREATE TABLE DEPARTMENT
  (DEP_NUM INTEGER NOT NULL,
    DEP_NAME VARCHAR(20),
    DEP_HEAD INTEGER,
    BUDGET DECIMAL(8,2) DEFAULT 0,
    LOCATION VARCHAR(20),
    PHONE_NUM INTEGER
);
```

Таблица «Проект» (PROJECT) содержит следующие атрибуты:

Pr_ID – наименование проекта, строка переменной длины с максимальной длиной 80 символов;

Pr_Name – описание проекта,

Pr_Descr – номер служащего

Leader – руководителя проекта, целочисленное, не может принимать значение NULL;

Budget – бюджет проекта, значение от 0 до 999999,99.

Таблица «Проект»

PR_ID	PR_NAME	PR_DESCR	LEADER	BUDGET
1112	Data based …		113	20000,00
1114	Parallel algorithms …		112	50000,0
2222	Knowledge …		224	100000,0

Следующая инструкция создает указанную таблицу:

```
CREATE TABLE PROJECT
(PR_ID INTEGER NOT NULL,
PR_NAME VARCHAR(80),
PR_DESCR VARCHAR(1000),
LEADER INTEGER,
BUDGET DECIMAL(8,2) DEFAULT 0
);
```

Таблица «Проект-Служащий» (PROJECT- EMPLOYEE) содержит следующие атрибуты:

Emp_Num – номер служащего, целочисленное, не может принимать значение NULL;

Pr_ID – код проекта, целочисленное, не может принимать значение NULL;

Data_S – содержит дату начала работ;

Data_F – содержит дату сдачи проекта.

Таблица «Проект-Служащий»

PR_ID	EMP_NUM	DATA_S	DATA_F
1112	111	25.08.2004	2.04.2005
1112	112	12.06.2004	22.05.2005
2222	111	22.10.2004	22.01.2005
2222	113	2.08.2004	22.08.2005

Следующая инструкция создает указанную таблицу:

```
CREATE TABLE PROJECT_ EMPLOYEE
 (PR_ID INTEGER NOT NULL,
     EMP_NUM INTEGER NOT NULL,
  DATA_S DATE,
  DATA_F DATE
);
```

INSERT добавляет одну или более новых строк данных к существующей таблице. Синтаксис оператора INSERT приведен в приложении Б. Значения вставляются в столбцы строки по порядку их следования, если не задан факультативный список столбцов. Если список столбцов задан на множестве всех доступных столбцов, то во все не перечисленные столбцы автоматически вставляется или значение по умолчанию, или NULL.

Если факультативный список столбцов упущен, предложение VALUES должно содержать значения для всех столбцов таблицы.

Чтобы вставить одну строку данных, должно присутствовать предложение VALUES и содержать определенный список значений.

Чтобы вставить несколько, строк данных, определите <*select_expr*>, которое возвращает уже существующие данные из другой таблицы. Выбранные строки должны соответствовать списку столбцов.

Предупреждение: Допустимо выбирать из той же таблицы, в которую строки вставляются, но эта практика не рекомендуется, так как подобные действия могут привести к бесконечным вставкам строк (зацикливанию).

Примеры использования оператора *INSERT*

1. Следующая инструкция добавляет строку в таблицу «Отдел»

```
INSERT INTO DEPARTMENT
VALUES (20, 'construct', 164, 'fl 6-7',456701);
```

2. Следующая инструкция добавляет строку в таблицу «Служащий», присваивает значения шести столбцам:

```
INSERT INTO EMPLOYEE (EMP_NUM,L_NAME,N_NAME,DEP_NUM,SALARY,JOB_CODE)
                VALUES (112,'Petrov','Stepan',10,1456.96,'admin');
```

3. Следующая инструкция определяет значения, чтобы вставить в таблицу, используя инструкцию SELECT:

```
INSERT INTO PROJECTS
  SELECT * FROM NEW_PROJECTS
    WHERE NEW_PROJECTS.START_DATE > '6-JUN-1994';
```

SELECT возвращает данные из таблицы, представления или сохраненной процедуры. Синтаксис оператора SELECT приведен в приложении Б. Различные инструкции SELECT выполняют следующие действия:

– возвращают одиночную стоку или часть строки из таблицы. Это операция упоминается, как sigleton выбор;

– непосредственно возвращают список строк или список частичных строк из таблицы;

– возвращают связанные строки, или частичные строки из JOIN двух или более таблиц;

– возвращают все строки, или частичные строки из UNION двух или более таблиц.

Любая инструкция SELECT содержит два обязательных предложения (SELECT, FROM) и возможно другие предложения (WHERE, GROUP BY, HAVING, UNION, PLAN, ORDER BY). Предложения SELECT и FROM обязательны и для sigleton, и для multi-row SELECT; все другие предложения перечисленные ниже факультативны. Назначение каждого предложения оператора SELECT приведено в таблице 1.1.

Таблица 1.1 – Назначение предложений, входящих в оператор SELECT

Предложение	Назначение
SELECT	Список столбцов, которые возвращаются.
FROM	Определяет таблицы, в которых ищутся значения.
WHERE	Определенное условие поиска, которое используется, чтобы выбрать необходимые стоки из множества всех строк. Предложение WHERE может содержать инструкцию SELECT, которая упоминается, как подзапрос
GROUP BY	Группирует возвращенные строки, основываясь на общих значениях столбцов. Используется совместно с HAVING.
HAVING	Определяет условие поиска для группы записей. Строки отбираются из значений столбцов, указанных в GROUP BY и значений агрегатных функций, вычисленных для каждой группы, образованной GROUP BY.
UNION	Комбинирует результаты двух или более инструкций SELECT, создавая одиночную динамическую таблицу, исключая повторяющиеся строки.
ORDER BY	Определяет порядок сортировки строк возвращенных SELECT, по умолчанию в возрастающем порядке (ASC), или в убывающем порядке (DESC).
PLAN	Определяет план запроса, который будет использоваться оптимизатором запроса вместо обычного выбора.

Примеры использования оператора SELECT

1. Следующая инструкция выбирает столбцы из таблицы «Отдел»:

```
SELECT DEP_NAME, BUDGET, PHONE_NUM
FROM DEPARTMENT;
```

2. Следующая инструкция использует шаблон *, что бы выбрать все столбцы и строки из таблицы «Проект»:

```
SELECT *
FROM PROJECT;
```

3. Следующая инструкция использует агрегатную функцию, чтобы сосчитать все строки в таблице, которые удовлетворяют условиям поиска определенным в предложении WHERE, количество проектов с бюджетом выше 50000:

```
SELECT COUNT(*)
FROM PROJECT
WHERE BUDGET > 50000;
```

4. Следующая инструкция использует составное условие поиска, определенное в предложении WHERE, позволяет определить фамилии и должности сотрудников 10 отдела, имеющих оклад выше 600:

```
SELECT L_NAME, STATE
FROM EMPLOYEE
WHERE DEP_NUM=10 AND SALARY>600;
```

5. Следующая инструкция выбирает два столбца и сортирует возвращенные строки по второму столбцу:

```
SELECT L_NAME, STATE
FROM EMPLOYEE
ORDER BY STATE;
```

6. Следующая инструкция позволяет оценить в каких проектах служащий принимает участие и вывести информацию с учетом даты окончания служащим работ по проекту

```
SELECT PR_NAME, EMP_NUM, DATA_F
FROM PROJECT_EMPLOYEE
WHERE EMP_NUM=112
ORDER BY DATA_F;
```

7. Следующая инструкция использует в условии поиска конструкцию IN (входит), позволяет определить наименования проектов, над которыми работают служащие номера, которых перечислены в списке:

```
SELECT PR_NAME
FROM PROJECT_EMPLOYEE
WHERE EMP_NUM IN(111,113);
```

1.3 Ход работы

4. Ознакомиться с описанием организации работы реляционных БД.
5. Создать тестовую БД, в соответствии с п. 1.2.3.
6. В соответствии, с п. 1.2.3:
   - создать тестовую таблицу,
   - занести в таблицу пять кортежей,
   - просмотреть содержимое таблицы.
7. Определить номер своего варианта. Номер варианта(x) определяется как x = N mod 20, где N – номер студента в группе.
8. По номеру своего варианта определить имя таблицы, которую надо будет создать при выполнении работы (Приложение А).
9. Ознакомиться с описанием базовых операторов для работы с таблицами.
10. Создать таблицу. Особое внимание надо уделить описанию первичного ключа, значений по умолчанию, описателям NOT NULL и конструкции CHECK.
11. Занести в таблицу образцы данных оператором INSERT INTO. Необходимо занести не менее 10 строк. Внимание! После того, как в таблицу занесены образцы данных, менять структуру таблицы можно только оператором ALTER TABLE.
12. Создать запрос, выводящий все строки таблицы.
13. Создать запрос, задающий порядок столбцов, отличный от исходного.
14. Продемонстрировать действие модификатора DISTINCT.
15. Ограничить вывод запроса, используя WHERE с простым условием.
16. Ограничить вывод запроса, используя WHERE и составное условие.
17. Продемонстрировать действие специальных функций IN, BETWEEN, LIKE и IS NULL в условии.
18. Продемонстрировать работу специальных функций с условием NOT.

1.4 Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.

2. Отчет должен содержать описание действий студента по конкретному варианту.
3. Обязательно должны быть указаны фактические значения, вводимые в диалоговые окна.
4. Отчет должен содержать:
   - таблицу исходных данных;
   - тексты запросов;
   - результаты их выполнения.

1.5 Контрольные вопросы

1. Объясните отличие архитектуры superserver и classic.
2. Какая информация указывается при создании базы данных?
3. Базовое понятие многопользовательских систем «транзакция».
4. Модели данных. Пояснить понятие реляционной модели данных.
5. Пояснить назвачение СУБД.
6. Основные понятие реляционных БД: отношение, атрибут, кортеж, домен, схема отношения.
7. Правила задания таблиц.
8. Ключевой атрибут, первичный ключ.
9. Требования к вводу данных оператором INSERT INTO.

ЛАБОРАТОРНАЯ РАБОТА № 2
SQL. Агрегатные функции

2.1 Цель работы

Изучение возможности обработки данных с помощью агрегатных функций языка SQL.

2.2 Основные положения

Запросы могут производить обобщенное групповое значение полей точно также, как и значение одного поля. Это делает с помощью агрегатных функций.

Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение. За исключением COUNT, не учитывают значения NULL. Часто используются совместно с предложением GROUP BY.

Агрегатные функции могут быть использованы в качестве выражений только в следующих случаях:

- список выбора инструкции SELECT (вложенный или внешний запрос);
- предложение HAVING.

Агрегатные функции используются подобно именам полей в предложении SELECT запроса, но с одним исключением, они берут имена поля как аргументы. Только числовые поля могут использоваться с SUM и AVG. С COUNT, MAX, и MIN, могут использоваться и числовые или символьные поля. Когда они используются с символьными полями, MAX и MIN будут транслировать их в эквивалент ASCII, который должен сообщать, что MIN будет означать первое, а MAX последнее значение в алфавитном порядке.

Порядок обработки предложений в операторе SELECT:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

2.2.1 Агрегатные функции

Агрегатные функции – это функции, которые работают не с одним значением, взятым из строки таблицы, а с группой значений.

Общий алгоритм, по которому работают агрегатные функции следующий:

1) производится упорядочивание таблицы по тем полям, по которым осуществляется группировка;

2) от каждой сформированной группы вычисляется требуемое значение, которое и заносится в результат.

Существуют следующие агрегатные функции:

- COUNT считает количество строк, которые вернул запрос;
- SUM суммирует значение всех полей, по указанному атрибуту;
- AVG находит среднее значение поля, по указанному атрибуту;
- MAX находит максимальное значение поля, по указанному атрибуту;
- MIN находит минимальное значение поля, по указанному атрибуту.

COUNT() – количество значений в группе

```
COUNT({[ALL | DISTINCT] <expr> | *})
```

expr – выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию. Агрегатные функции в качестве выражения не допускаются.

Функция COUNT возвращает количество значений в группе, которые не являются NULL. При указании DISTINCT из выборки устраняются дубликаты, ALL является значением

по умолчанию для всех выборки значений не NULL. Если вместо выражения *expr* указана звёздочка (*), то будут подсчитаны все строки. Функция COUNT(*) не принимает параметры и не может использоваться с ключевым словом DISTINCT. Для функции COUNT(*) не нужен параметр *expr*, так как по определению она не использует сведения о каких-либо конкретных столбцах. Функция COUNT(*) возвращает количество строк в указанной таблице, не отбрасывая дублированные строки. Она подсчитывает каждую строку отдельно. При этом учитываются и строки, содержащие значения NULL. Для пустой выборки данных или если при выборке окажутся одни значения, содержащие NULL, функция возвратит значение равное 0.

SUM() – сумма элементов выборки

```
SUM([ALL | DISTINCT] <expr>)
```

expr – выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию. Агрегатные функции в качестве выражения не допускаются.

Функция SUM возвращает, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. ALL является опцией по умолчанию. При ней обрабатываются все значения из выборки, не содержащие NULL. При указании DISTINCT из выборки устраняются дубликаты, после осуществляется подсчёт.

AVG() – среднее значение для группы

```
AVG([ALL | DISTINCT] <expr>)
```

expr – выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию. Агрегатные функции в качестве выражения не допускаются.

Функция AVG возвращает среднее значение для группы. Значения NULL пропускаются. Параметр ALL применяет агрегатную функцию ко всем значениям. ALL является параметром по умолчанию. Параметр DISTINCT указывает на то, что функция AVG будет выполнена только для одного экземпляра каждого уникального значения, независимо от того, сколько раз встречается это значение. В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

MAX() – максимальный элемент выборки

```
MAX([ALL | DISTINCT] <expr>)
```

expr – выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию. Агрегатные функции в качестве выражения не допускаются.

Функция MAX возвращает максимальный элемент выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. Если аргумент функции строка, то функция вернёт значение, которое окажется последним в сортировке при применении COLLATE.

Примечание. Параметр DISTINCT не имеет смысла при использовании функцией MAX и доступен только для совместимости со стандартом.

MIN() – минимальный элемент выборки

```
MIN([ALL | DISTINCT] <expr>)
```

expr – выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию. Агрегатные функции в качестве выражения не допускаются.

Функция MIN возвращает, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. Если аргумент функции строка, то функция вернёт значение, которое окажется первым в сортировке при применении COLLATE.

Примечание. Параметр DISTINCT не имеет смысла при использовании функцией MIN и доступен только для совместимости со стандартом.

2.2.2 Тестовая база данных

Рассмотрим работу агрегатных функций на примере базы данных, состоящий из трех таблиц – «Торговый агент», «Покупатель», «Сделка».

Таблица «Торговый агент» *(*Salespeople) содержит следующие атрибуты:

- SNUM – номер агента;
- SNAME – имя агента;
- CITY – город, где работает агента;
- COMM – комиссионные торгового агента.

Таблица «Торговый агент»

SNUM	SNAME	CITY	COMM
1001	Peel	London	0.12
1002	Serres	San Jose	0.13
1004	Motika	London	0.11
1007	Rifkin	Barcelona	0.15
1003	Axelrod	New York	0.10

Таблица «Покупатель»

CNUM	CNAME	CITY	RATING	SNUM
2001	Hoffman	London	100	1001
2002	Giovanni	Rome	200	1003
2003	Liu	SanJose	200	1002
2004	Grass	Berlin	300	1002
2006	Clemens	London	100	1001
2008	Cisneros	SanJose	300	1007
2007	Pereira	Rome	100	1004

Таблица «Покупатель» (Customers) состоит из следующих атрибутов:

- CNUM – номер покупателя;
- CNAME – имя покупателя;
- CITY – город проживания покупателя;
- RATING – некоторый рейтинг, присвоенный покупателю;
- SNUM – номер торгового агента, за которым закреплен покупатель.

Таблица «Сделка» (Orders) содержит следующие атрибуты:

- ONUM – номер сделки;
- AMT – сумма сделки;
- ODATE – дата свершения сделки;
- CNUM – номер покупателя;
- SNUM – номер торгового агента.

Таблица «Сделка»

ONUM	AMT	ODATE	CNUM	SNUM
3001	18.69	10/03/1990	2008	1007
3003	767.19	10/03/1990	2001	1001
3002	1900.10	10/03/1990	2007	1004
3005	5160.45	10/03/1990	2003	1002
3006	1098.16	10/03/1990	2008	1007
3009	1713.23	10/04/1990	2002	1003
3007	75.75	10/04/1990	2004	1002
3008	4723.00	10/05/1990	2006	1001
3010	1309.95	10/06/1990	2004	1002
3011	9891.88	10/06/1990	2006	1001

2.2.3 Действия с базовыми агрегатными функциями

Чтобы найти сумму всех покупок в таблице «Сделки», можно ввести запрос, представленный ниже, результат выполнения которого – число 26658.4:

```
SELECT SUM(amt) FROM Orders;
```

Запрос, представленный ниже, вернет среднее значение сумм сделок – 2665.84.

```
SELECT AVG(amt) FROM Orders;
```

Иногда возникает необходимость решить обратную задачу – подсчитать количество значений поля вместе с повторениями. Для этого существует описатель **ALL** (подразумевается по умолчанию).

Например, необходимо определить количество торговых агентов snum с учетом повторений. Результат – 7.

```
SELECT COUNT (ALL snum) FROM Customers;
```

Чтобы подсчитать общее число строк в таблице, используется функция COUNT со звездочкой вместо имени поля, как, например, в следующем примере. Результат – 7.

```
SELECT COUNT(*) FROM Customers;
```

Внимание! Только **COUNT**(*) может подсчитывать значения NULL. Все остальные функции игнорируют неопределенные значения.

2.2.4 Простые вычисления

Столбцы, значение которых может быть получено с помощью простых арифметических действий через другие столбцы в БД, как правило, не хранятся. SQL предоставляет простой способ производить подобные вычисления. Также можно помещать в некоторый столбец константу.

Например, чтобы представить комиссионные торгового агента в процентном отношении, а не в виде десятичного числа можно записать такой запрос:

```
SELECT snum, sname, city, comm*100 FROM Salespeople;
```

Можно дополнить указанный запрос знаком процента, выводимым после поля comm*100:

```
SELECT snum, sname, city, comm*100, ' % ' FROM Salespeople;
```

Знак процента в данном случае – константное выражение. При выполнении вычислений в запросе допустимы арифметические действия – сложение, вычитание, умножение, деление.

2.2.5 Вычисления в агрегатных функциях

Допустим, необходимо вывести 10% от стоимости самой дорогой сделки. Тогда можно записать такой запрос:

```
SELECT MAX (amt*0.1) FROM Orders;
```

Таким образом, можно выполнять вычисления с использованием других полей и арифметических действий.

Внимание! Вложение агрегатов не допускается.

2.2.6 Использование GROUP BY

В случае если таблица рассматривается целиком, в списке вывода запроса присутствуют только агрегатные функции.

В случае если производится группировка таблицы по какому-то полю, в списке вывода могут присутствовать те поля, относительно которых таблица группируется и агрегатные функции. При этом все поля, не охваченные агрегатными функциями, **должны быть перечислены в предложении GROUP BY**.

Предположим, что необходимо найти сделку с максимальной стоимостью для каждого торгового агента. Можно сделать персональный запрос для каждого из них, выбрав MAX(amt) из таблицы сделок. Можно записать следующий запрос:

```
SELECT snum, MAX(amt) FROM Orders GROUP BY snum;
```

Результат выполнения запроса:

snum	
1001	9891.88
1002	5160.45
1003	1713.23
1004	1900.10
1007	1098.16

В данном случае таблица группируется относительно номера торгового агента, поэтому поле snum присутствует в списке вывода запроса и в предложении GROUP BY.

2.2.7 Использование HAVING

Так же, как и предложение WHERE ограничивает строки в наборе данных теми, которые удовлетворяют условию поиска, с той разницей, что предложение HAVING накладывает ограничения на агрегированные строки сгруппированного набора.

Предложение HAVING не является обязательным и может быть использовано только в сочетании с предложением GROUP BY.

Условие(я) в предложении HAVING может ссылаться на:

- любой агрегированный столбец в списке выбора SELECT. Это наиболее широко используемый случай;
- любое агрегированное выражение, которое не находится в списке выбора SELECT, но разрешено в контексте запроса. Иногда это полезно;
- любой столбец в списке GROUP BY. Однако более эффективно фильтровать не агрегированные данные на более ранней стадии в предложении WHERE;
- любое выражение, значение которого не зависит от содержимого набора данных (например, константа или контекстная переменная). Это допустимо, но совершенно бессмысленно, потому что такое условие, не имеющее никакого отношения к самому набору данных, либо подавит весь набор, либо оставит его не тронутым.

Предложение HAVING не может содержать:

- не агрегированные выражения столбца, которые не находятся в списке GROUP BY;
- позицию столбца. Целое число в предложении HAVING – просто целое число;
- псевдонимы столбца – даже, если они появляются в предложении GROUP BY.

Например, если необходимо узнать, какие торговые агенты имеют заработок от одной сделки более чем 3000, и в какой день, можно использовать запрос вида:

```
SELECT snum, odate, MAX(amt)
FROM Orders
GROUP BY snum, odate
HAVING MAX(amt) > 3000.00;
```

Результат работы запроса:

Snum	Odate	
1001	10/05/1990	4723.00
1001	10/06/1990	9891.88
1002	10/03/1990	5160.45

2.2.8 Выражение ROWS

```
SELECT ...
FROM ...
...
ROWS m [TO n]
```

где *m*, *n* – любые целочисленные выражения.

Выражение ROWS принимает все типы целочисленных (integer) выражений в качестве аргумента – без скобок. Скобки могут требоваться для правильных вычислений внутри выражения, и вложенный запрос также должен быть обернут в скобки. Вызов ROWS *m* приведёт к возвращению первых *m* записей из набора данных.

Особенности при использовании ROWS с одним аргументом:

- если *m* больше общего числа записей в возвращаемом наборе данных, то будет возвращён весь набор данных;
- если *m* = 0, то будет возвращён пустой набор данных;
- если *m* < 0, выдаётся ошибка.

В случае указания ROWS *m* TO *n*, то будут возвращены записи с *m* по *n* из набора данных.

Важно. Нумерация записей в наборе данных начинается с 1.

Особенности при использовании ROWS с двумя аргументами:

- если *m* больше общего количества строк в наборе данных и *n* >= *m*, то будет возвращён пустой набор данных;
- если число *m* не превышает общего количества строк в наборе данных, а *n* превышает, то выборка ограничивается строками, начиная с *m* до конца набора данных;
- если *m* < 1 и *n* < 1, то оператор SELECT выдаст ошибку;
- если *n* = *m* -1, то будет возвращён пустой набор данных;
- если *n* < *m* -1, то оператор SELECT выдаст ошибку.

В сущности, ROWS заменяет собой нестандартные выражения FIRST и SKIP, за исключением единственного случая, когда указывается только SKIP, т.е. когда возвращается весь набор данных за исключением пропуска указанного числа записей с начала.

Для того, что реализовать такое поведение с помощью ROWS, необходимо указать второй аргумент, заведомо больший, чем размер возвращаемого набора данных. Или запросить число записей в возвращаемом наборе через подзапрос.

Нельзя использовать ROWS вместе с FIRST/SKIP в одном и том же операторе SELECT, но можно использовать разный синтаксис в разных подзапросах.

При использовании ROWS с выражением UNION, он будет применяться к объединённому набору данных, и должен быть помещён после последнего SELECT.

При необходимости ограничить возвращаемые наборы данных одного или нескольких операторов SELECT внутри UNION, можно воспользоваться следующими вариантами:

1) использовать FIRST/SKIP в этих операторах SELECT. Необходимо помнить, что нельзя локально использовать выражение ORDER BY в SELECT внутри UNION – только глобально, ко всему суммарному набору данных;

2) преобразовать SELECT в производные таблицы с выражениями ROWS.

Следующий запрос вернёт первые 10 имён из таблицы PEOPLE

```
SELECT id, name FROM People ORDER BY name ASC ROWS 1 TO 10
```

или его эквивалент

```
SELECT id, name FROM People ORDER BY name ASC ROWS 10
```

Следующий запрос вернёт все записи из таблицы PEOPLE, за исключением первых 10 имён:

```
SELECT id, name FROM People ORDER BY name ASC
ROWS 11 TO (SELECT COUNT(*) FROM People)
```

А этот запрос вернёт последние 10 записей (обратите внимание на скобки):

```
SELECT id, name FROM People ORDER BY name ASC
ROWS (SELECT COUNT(*) - 9 FROM People)
     TO (SELECT COUNT(*) FROM People)
```

Запрос, представленный ниже, вернёт строки 81-100 из таблицы PEOPLE:

```
SELECT id, name FROM People ORDER BY name ASC
ROWS 81 TO 100
```

И FIRST/SKIP, и ROWS могут быть использованы без выражения ORDER BY, хотя это редко имеет смысл, за исключением случая, когда необходимо быстро взглянуть на данные таблицы – получаемые строки при этом будут чаще всего в случайном порядке. В этом случае запрос вроде «`SELECT * FROM TABLE1 ROWS 20`» вернёт 20 первых записей, а не целую таблицу (которая может очень большой).

2.2.9 Вычисляемые поля COMPUTED BY

Вычисляемые поля могут быть определены с помощью предложения **COMPUTED [BY]** или **GENERATED ALWAYS AS**. Они эквивалентны по смыслу. Для вычисляемых полей не требуется описывать тип данных (но допустимо), СУБД вычисляет подходящий тип в результате анализа выражения. В выражении требуется указать корректную операцию для типов данных столбцов, входящих в его состав. При явном указании типа столбца для вычисляемого поля результат вычисления приводится к указанному типу.

```
CREATE TABLE SALARY (
ID INTEGER NOT NULL,
DATA DATE DEFAULT 'NOW' NOT NULL,
UPD_ID VARCHAR(5) NOT NULL,
SALARY DECIMAL(8,2) NOT NULL,
CHANGE DECIMAL(8,2) DEFAULT 0 NOT NULL,
SALARY_CHANGE DECIMAL(8,2) GENERATED ALWAYS AS (SALARY * CHANGE /
100),
NEW_SALARY DECIMAL(8,2) COMPUTED BY (SALARY + SALARY * CHANGE /
100));

CREATE TABLE STUDENT (
id INTEGER NOT NULL PRIMARY KEY,
name VARCHAR(10),
birthdate DATE,
age COMPUTED BY (DATEDIFF(DAY,birthdate,current_date)/365));
```

2.3 Порядок выполнения лабораторной работы

1. Ознакомиться с принципами работы агрегатных функций COUNT, SUM, AVG, MAX, MIN.
2. Продемонстрировать использование COUNT(*)
3. Продемонстрировать выполнение простых вычислений в запросе.
4. Использовать простое вычисление как параметр агрегатной функции.
5. Ознакомиться с использованием предложения GROUP BY, продемонстрировать его работу.
6. Ознакомиться с использованием предложения HAVING, продемонстрировать его работу.
7. Составить и защитить отчет.

2.4 Задание на работу

Вариант задания, соответствует варианту, полученному в Лабораторной работе №1.

2.5 Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.
2. Отчет должен содержать таблицу исходных данных, тексты запросов и результаты их выполнения.

2.6 Контрольные вопросы

1. Назначение агрегатных функций.
2. Возможности предложения GROUP BY.
3. Условия отбора групп. Предложения HAVING и WHERE назначение и отличия в использовании.
4. Простые вычисления над данными.
5. Требования к списку выводимых столбцов фразы SELECT при задании группировки таблицы по какому-либо полю.

ЛАБОРАТОРНАЯ РАБОТА № 3
Создание схемы базы данных. Ссылочная целостность

3.1 Цель работы

Научиться анализировать предметную область с целью создания схемы БД, учитывая ссылочную целостность набора.

3.2 Основные положения

3.2.1 Типы связей между отношениями. Требования к ссылочной целостности данных

Свое название реляционные базы данных получили именно по той причине, что таблицы в БД не существуют независимо друг от друга. Таблицы взаимосвязаны друг с другом, т.е. действие, произведенное в одной таблице, вызовет некоторые действия в другой таблице.

Существует три основных класса связей между таблицами:

- один к одному (1:1);
- один ко многим (1:М);
- многие ко многим (М:М).

На практике связи первого типа используются редко. Связи третьего типа не реализуются в РБД напрямую, одну связь многие ко многим приводят к двум связям один ко многим. Поэтому связь 1:М используется наиболее часто, ее следует рассмотреть наиболее подробно.

Пусть имеется две таблицы следующего вида:

1. Клиент (Номер_клиента, ФИО_клиента, Счет_клиента).
2. Продажи (Номер_клиента, Наименование_товара, Количество_товара).

Таблицы связанны отношением 1:М, т.е. один клиент может совершить много покупок, каждая покупка совершается только одним клиентом. В таблицах хранятся следующие данные:

Таблица «Клиент»

Номер_клиента	**ФИО_клиента**	**Счет_клиента**
1	Сидоров И.И.	1923563
2	Петров А.Б.	1736523
3	Федоров Н.Г.	9265623

Таблица «Продажи»

Номер_клиента	**Наименование_товара**	**Количество_товара**
1	Шоколад «Корона»	2
1	Шейки Куринные	1,5
2	Крупа манная	3

Очевидно, что поле Номер_клиента в обеих таблицах имеет одинаковый смысл - оно обозначает номер, присвоенный клиенту. В случае если таблицы не имеют связи друг с другом, ничего не мешает добавить в таблицу «Продажи» запись, например, (8, «Хлеб белый»,1), несмотря на то, что в таблице «Клиент» нет покупателя с номеров 8. Это нарушение целостности данных, так как такая строка может быть занесена, но она не соответствует действительности. Чтобы подобная ситуация стала невозможной, необходимо установить связь между таблицами по полю «Номер_клиента». В этом случае сервер БД автоматически проследит, чтобы поле «Номер_клиента» из вставляемой строки существовало в таблице «Клиенты».

Подобной проверки достаточно, чтобы условие целостности данных выполнялось при добавлении данных в таблицы. Но его недостаточно при манипулировании данными.

Рассмотрим ситуацию, когда необходимо удалить из таблицы «Клиент» некоторую запись, например, клиента с номером 1. В случае если таблицы не связанны, удаление клиента повлечет за собой изменение только одной таблицы. В таблице «Продажи» останутся сведения

о покупках покупателя с номером 1. Такая ситуация - также нарушение целостности данных, так как о данном покупателе, после его удаления, базе данных ничего не известно. В случае если таблицы связанны, удаление покупателя может повлечь за собой удаление всех его покупок (говорят, что удаление каскадируется). Использование конструкций оператора CREATE TABLE для обеспечения целостности данных приведено в приложении В.

Такая же ситуация с модификацией данных в родительской таблице. Если покупатель с номером 1 изменил номер на 5, то в связанных таблицах изменение родительской таблицы повлечет за собой автоматическое изменение дочерних таблиц.

Кроме арности связи и ограничений целостности различают также идентифицирующие и не идентифицирующие связи.

С помощью идентифицирующей связи устанавливается взаимосвязь между зависимыми сущностями (клиент и его покупки). Не идентифицирующая связь устанавливает взаимосвязь между независимыми сущностями (автомобиль и его цвет). Разница между идентифицирующей и не идентифицирующей связью состоит в том, что при идентифицирующей связи внешний ключ является частью первичного ключа дочерней сущности, а при не идентифицирующей связи внешний ключ не входит в первичный ключ дочерней сущности. На **ER-диаграмме** идентифицирующая связь изображается непрерывной линией, не идентифицирующая – пунктирной.

3.2.2 Стандартная нотация ER-диаграмм

Приведем пояснения к стандартной нотации ER-диаграмм, в которой представлены варианты к лабораторной работе.

Сущности. Сущность изображается прямоугольником, над которым пишется имя сущности. Одна сущность соответствует одной таблице. Прямоугольник разделяется линией на две части. В верхней части указываются ключевые атрибуты. После имени атрибута могут стоять символы, уточняющие назначение атрибута. Допустимые символы:

- PK – первичный ключ;
- AK – альтернативный ключ;
- FK – внешний ключ;
- IE – inversion entry, говорит о том, что по данному полю должен быть создан индекс.

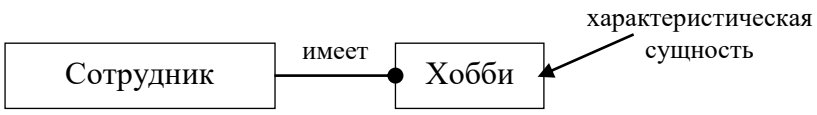
Существует два типа сущностей:

– *зависимая сущность (dependent entity)* – для определения экземпляра такой сущности необходимо сослаться на экземпляр независимой сущности, с которой связана зависимая сущность. Пример: сущности – заказ и позиция заказа. Для идентификации позиции заказа нужно сослаться на заказ, в который входит данная позиция;

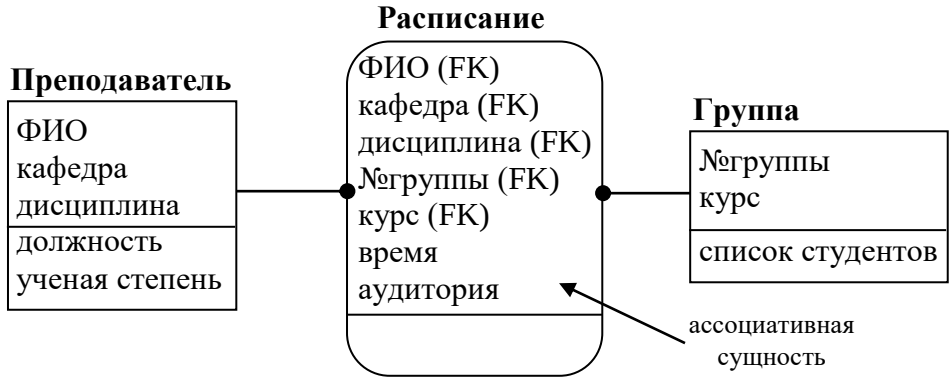
– *независимая сущность (independent entity)* – для определения экземпляра сущности нет необходимости ссылаться на другие сущности. Пример: сущности – заказ и позиция заказа. Для определения заказа (сущности) нет необходимости ссылаться на позиции этого заказа (другие сущности).

Типы зависимых сущностей:

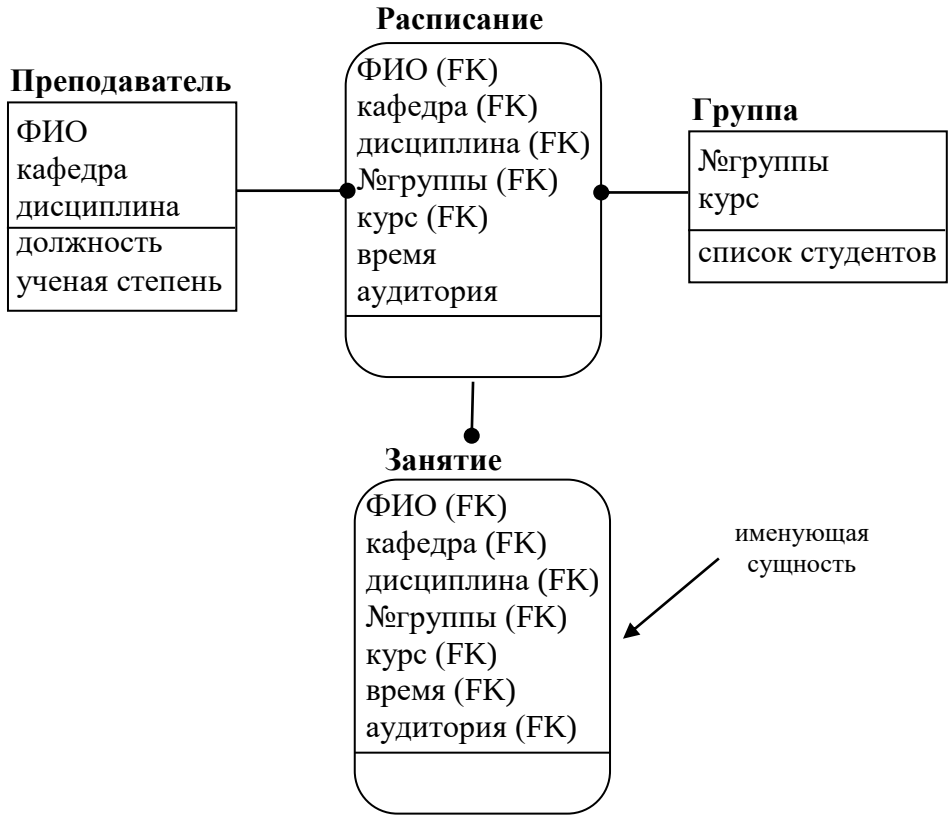
- *характеристическая* – это зависимая дочерняя сущность, которая связана только с одной родительской сущностью и по смыслу хранит информацию о характеристиках родительской сущности. Единственная цель характеристики в рамках рассматриваемой предметной области состоит в описании или уточнении некоторой другой сущности. Необходимость в них возникает в связи с тем, что сущности реального мира имеют иногда многозначные свойства. Муж может иметь несколько жен, книга – несколько характеристик переиздания (исправленное, дополненное, переработанное, ...) и т.д. Существование характеристики полностью зависит от характеризуемой сущности: женщины лишаются статуса жен, если умирает их муж;



– *ассоциативная* – сущность, связанная с несколькими родительскими сущностями. Такая сущность содержит информацию о связях сущности;



– *именующая* – частный случай ассоциативной сущности, не имеет собственных атрибутов, только атрибуты родительской сущности, мигрировавших в качестве внешнего ключа

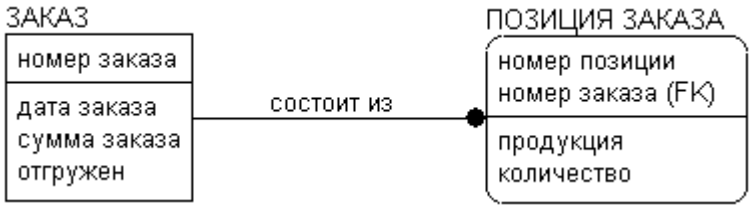


Связи. Связь между двумя отношениями изображается с помощью линии. Идентифицирующая связь изображается сплошной линией, не идентифицирующая - пунктирной. Арность связи указывается следующим образом: со стороны «многие» ставится жирная точка, со стороны «один» точка не ставится. Допустимость Null -значений изображается ромбиком с той стороны связи, где позволяются Null – значения.

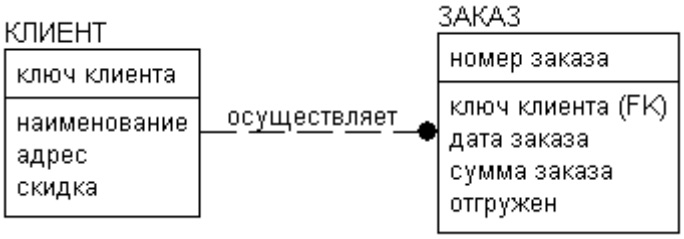
Существует несколько **видов связей**:

– *идентифицирующая (identifying relationship)* – указывает на то, что дочерняя сущность в связи является зависимой от родительской сущности, т.е. экземпляр зависимой сущности может быть однозначно определен, только если в этом экземпляре есть ссылка на экземпляр независимой сущности. В идентифицирующем отношении сущность-потомок всегда является зависимой от идентифицирующей сущности. Первичный ключ родительской сущности наследуется в качестве первичного ключа дочерней сущностью (внешний ключ).

Идентифицирующая связь отображается сплошной линией, причем дочерняя сущность является зависимой и поэтому отображается *прямоугольником со скругленными углами*;



– *неидентифицирующая (non-identifying relationship)* – показывает на зависимость между родительской и дочерней сущностями, при этом экземпляр дочерней сущности может быть однозначно идентифицирован без ссылки на экземпляр родительской сущности, т.е. родительская сущность связана с дочерней сущностью, но однозначно не определяет ее. Первичный ключ родительской сущности наследуется в качестве неключевого атрибута дочерней сущности. Неидентифицирующая связь отображается штриховой линией;



– *иерархическая* – указывает на связь сущности саму на себя. Если у родительской таблицы линия имеет белый ромб, то это означает, что внешний ключ дочерней таблицы может содержать нулевые значения. Если связь является иерархической рекурсией, то первичный ключ этой сущности мигрирует в область атрибутов этой же сущности. При этом внешнему ключу должно быть обязательно назначено имя роли. Связь «руководит/подчиняется» на рисунке 6 позволяет хранить древовидную иерархию подчиненности сотрудников. Такой вид рекурсивной связи называется иерархической рекурсией (hierarchical recursion) или рыбный крючок (fish hook) и задает связь, когда руководитель (экземпляр родительской сущности) может иметь множество подчиненных (экземпляров дочерней сущности), но подчиненный имеет только одного руководителя.

Категории. В процессе моделирования данных могут быть выявлены сущности, часть атрибутов и связей которых одинаковы. В этом случае используется *иерархия категорий*. Все общие атрибуты выделяются в сущность, называемую **супертипом**, а отличающиеся атрибуты помещаются в сущности – **подтипы**, связанные с супертипом. При помощи дискриминанта определяется, с экземпляром какого подтипа связан экземпляр супертипа.

Отношения **категоризации** – отношения между двумя и более сущностями, в которых каждый экземпляр одной сущности, называемой *общей*, связан в точности с одним экземпляром сущности, называемой *сущностью-категорией*. Категория выделяется из общей сущности по определенному *признаку* (дискриминанту). Дискриминатор (discriminator) – атрибут родового предка, который показывает, как отличить одну категориальную сущность от другой (атрибут «тип сотрудника» на рисунке 3.1).

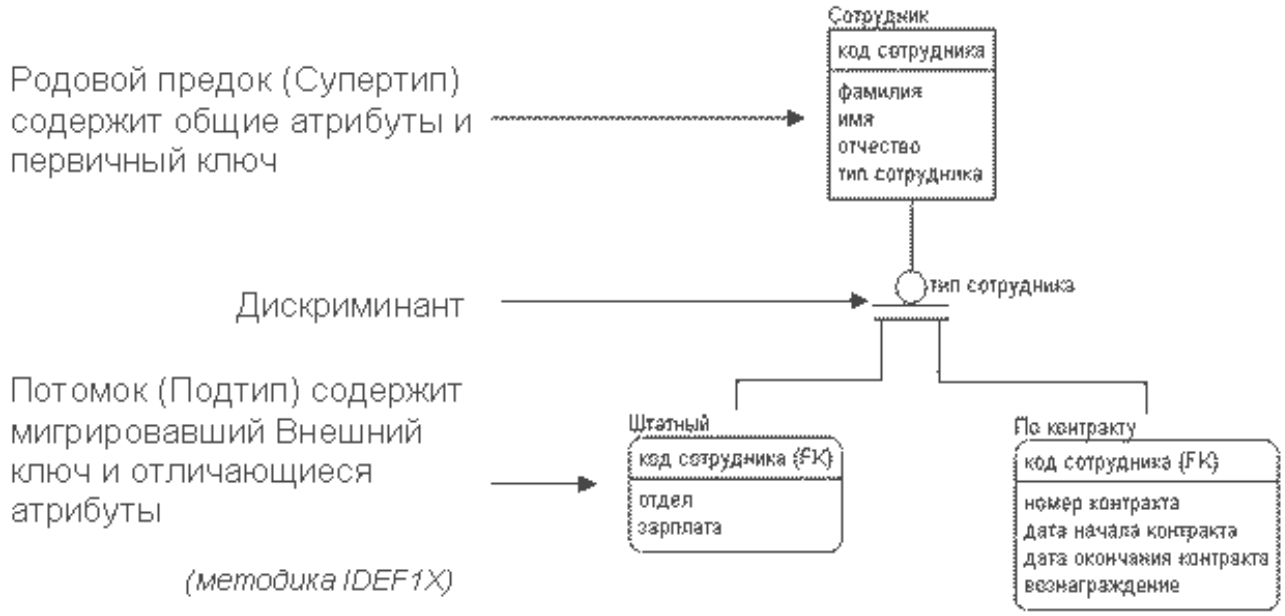


Рисунок 3.1 – Связь типа иерархия категорий в методике IDEF1X

Иерархия категорий может быть двух типов – полная и неполная. Определить, какой тип иерархии категорий представлен в модели данных, можно по стилю отображения дискриминанта: *дискриминант с двумя горизонтальными линиями* свидетельствует о *полноте* иерархии категорий, с *одной горизонтальной линией* – о ее *неполноте*. Полная иерархия категорий свидетельствует о завершенности анализа. В примере с сотрудником полная иерархия категорий говорит о том, что любой сотрудник является либо штатным сотрудником, либо работающим по контракту и никаких других вариантов быть не может. Если в примере изменить иерархию категорий на неполную, это будет свидетельствовать о том, что помимо сотрудников, зачисленных в штат и работающих по контракту, могут быть и другие типы сотрудников, но на данном этапе моделирования данных, они еще не выявлены.

Рисунок 3.2 – Виды иерархии категорий в методике IDEF1X

Правила отношений категоризации:

1) сущность типа «категория» может иметь только одну общую сущность;

2) сущность-категория, принадлежащая одному отношению категоризации, может быть общей сущностью в другом отношении категоризации;

3) сущность может являться общей в любом количестве отношений категоризации;

4) атрибуты первичного ключа сущности-категории должны совпадать с атрибутами первичного ключа общей сущности;

5) все экземпляры сущности-категории имеют одно и то же значение дискриминатора, следовательно, все экземпляры других категорий должны иметь другое значение дискриминатора.

Рассмотрим на примере использование понятия категории. Допустим, нам необходимо хранить следующую информацию о родителях и детях:

Родитель (Номер, ФИО, Адрес, Образование).
Ребенок (Номер, ФИО, Адрес, Номер_детского_садика).
ЯвляетсяРебенком(Номер_родителя, Номер_ребенка).

Схема данных:
Родитель – 1:M → Ребенок

Очевидно, что для ребенка поле «Образование» не имеет смысла, как и поле «Номер_детского_садика» для родителя. Кроме того, в случае, если подобную информацию хранить в виде двух таблиц, то возникает дублирование информации и проблемы с целостностью данных. Например, адрес ребенка совпадает с адресом родителей, и если данные вводятся вручную, то можно ввести адрес ребенка отличный от адреса родителя.

Подобная проблема решается следующим образом: вводится обобщающая категория «Личность», в нее выносятся общие поля из двух дочерних таблиц. Дочерние таблицы привязываются к родительской связью один к одному.

Пример:
Личность (Номер, ФИО);
Родитель (Номер, Адрес, Образование);
Ребенок (Номер, Номер_детского_садика);
ЯвляетсяРебенком(Номер_родителя, Номер_ребенка)

Схема данных:
Родитель –1:1→ Личность ←1:1– Ребенок

Для того чтобы добавить данные о ребенке, нужно занести данные в две таблицы – «личность» и «ребенок». Например:

```
ISERT INTO Личность VALUES (2,'Иванов И.И.');
ISERT INTO Ребенок VALUES (2,'д/с №23');
```

Нумерация в таблицах сквозная!

На рисунке 3.3 изображена схема категориальной связи.

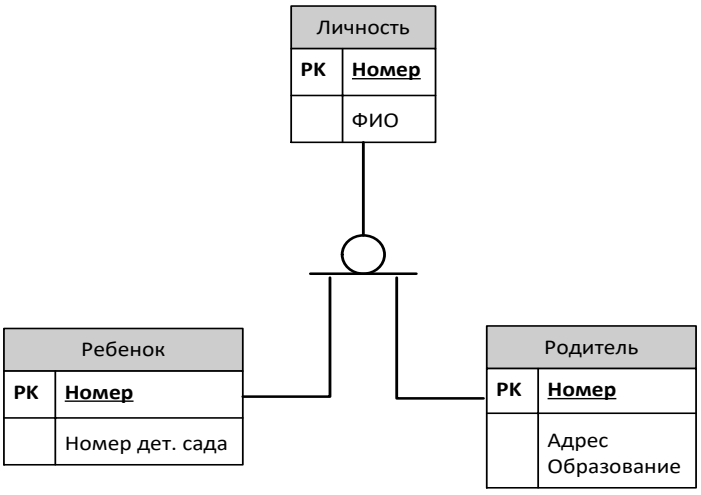


Рисунок 3.3 – Схеме категориальной связи

Допустимость NULL-значений связи на диаграмме указывается ромбиком.

Связь таблиц в реляционной БД устанавливается при создании таблиц с помощью описателя FOREIGEN KEY (см. приложение В).

3.2.3 Ограничения в SQL

Существуют четыре вида ограничений:

- первичный ключ (PRIMARY KEY);
- уникальный ключ (UNIQUE);
- внешний ключ (REFERENCES или FOREIGN KEY);
- проверочное ограничение (CHECK).

```
...
  ,[CONSTRAINT <constraint_name>] {
     PRIMARY KEY (<domain1>[,<domainN >]) [USING [ASC|DESC] INDEX
<index_name>]|
     FOREIGN KEY (<domain1>[,<domainN >])
         REFERENCES <table_name >(<domain1 > [,<domainN >])
         [ON UPDATE <rule >]
         [ON DELETE <rule >] [USING INDEX <index_name>] |
     UNIQUE (<domain1 >[,<domainN >]) [USING [ASC|DESC] INDEX <index_name >]
|
     CHECK <check_condition>
  }
...
```

Ограничения могут быть указаны на уровне столбца (ограничения столбцов) или на уровне таблицы (табличные ограничения). Ограничения уровня таблицы необходимы, когда ключи (ограничение уникальности, первичный ключ или внешний ключ) должны быть сформированы по нескольким столбцам, или, когда ограничение CHECK включает несколько столбцов, т.е. действует на уровне записи. Синтаксис для некоторых типов ограничений может незначительно отличаться в зависимости от того определяется ограничение на уровне столбца или на уровне таблицы.

Ограничение на уровне столбца указывается после определения других характеристик столбца и может включать только столбец указанный в этом определении.

Ограничения на уровне таблицы указываются после определений всех столбцов. Ограничения таблицы являются более универсальным способом записи ограничений, поскольку позволяют ограничение более чем для одного столбца таблицы.

Можно смешивать ограничения столбцов и ограничения таблиц в одном операторе CREATE TABLE.

Существует два *типа ограничений*:

- *именованные* – необязательное предложение CONSTRAINT задаёт имя ограничения;
- *автоматически создаваемые индексы* – для первичного ключа (PRIMARY KEY), уникального ключа (UNIQUE) и внешнего ключа (REFERENCES) система автоматически создаёт соответствующий индекс.

Если задано имя ограничения, то автоматически создаваемому индексу будет присвоено это имя. Если имя ограничения не задано, то ограничению будет присвоено имя, автоматически генерированное СУБД. Для любого ограничения можно указать имя.

Предложение USING позволяет задать определённое пользователем имя автоматически создаваемого индекса, и опционально определить, какой это будет индекс – по возрастанию (по умолчанию) или по убыванию.

Первичный ключ (PRIMARY KEY, PK). Ограничение первичного ключа PRIMARY KEY строится на поле с заданным ограничением NOT NULL и требует уникальности значений столбца. Таблица может иметь только один первичный ключ. Первичный ключ по единственному столбцу может быть определён как на уровне столбца, так и на уровне таблицы. Первичный ключ по нескольким столбцам может быть определён только на уровне таблицы.

```
CREATE TABLE COUNTRY (
     COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,
     CURRENCY VARCHAR(10) NOT NULL);
```

или (рекомендуемый вариант, с указанием имени CONSTRAINT)

```
CREATE TABLE MY_TABLE(
  ID INTEGER NOT NULL,
  SOME_ID INTEGER NOT NULL,
  ......
  CONSTRAINT PK_MY_TABLE PRIMARY KEY(ID)
);
```

Внешний ключ (FOREIGN KEY, FK)

```
FOREIGN KEY (  < domain1 >  [,< domainN >] )
  REFERENCES < TABLE_NAME > [  < Rdomain1 >  [,< RdomainN >] ]
  [ON DELETE < rule > ] [ON UPDATE < rule > ];
```

где

```
< rule > = {NO ACTION | CASCADE | SET DEFAULT | SET NULL };
```

На уровне столбца ограничение внешнего ключа определяется предложением REFERENCES. После ключевого слова REFERENCES указывается имя родительской таблицы, на первичный или уникальный ключ которой ссылается описываемый внешний ключ. Имя столбца родительской таблицы, являющегося первичным (уникальным) ключом, помещается сразу после имени таблицы и заключается в круглые скобки, например

```
... ,
ARTIFACT_ID INTEGER REFERENCES COLLECTION(ARTIFACT_ID),
...
```

Синтаксис определения внешнего ключа на уровне таблицы несколько отличается. После определения всех столбцов, с их ограничения уровня столбца, вы можете определить именованное ограничение внешнего ключа уровня таблицы, используя ключевые слова FOREIGN KEY и имён столбцов для которых оно применяется:

```
... ,
CONSTRAINT FK_ARTSOURCE FOREIGN KEY(DEALER_ID, COUNTRY)
REFERENCES DEALER (DEALER_ID, COUNTRY),
```

Внешний ключ не требует ограничения NOT NULL для столбца или столбцов, на которые ссылается внешний ключ, хотя оно будет, если внешний ключ ссылается на столбец или столбцы, входящие в первичный ключ родительской таблицы. Тем не менее, внешний ключ может ссылаться на столбец или столбцы, входящие в уникальный ключ. NULL допускается в уникальных ключах, и, с некоторыми ограничениями, в уникальных ключах, построенных на нескольких столбцах.

Для обеспечения дополнительной целостности данных можно указать необязательные опции ON DELETE и ON UPDATE, которые обеспечат согласованность данных между родительскими и дочерними таблицами по заданным правилам:

– предложение ON UPDATE определяет, что произойдёт с записями подчинённой таблицы при изменении значения первичного/уникального ключа в строке главной таблицы;

– предложение ON DELETE определяет, что произойдёт с записями подчинённой таблицы при удалении соответствующей строки главной таблицы.

Правила удаления управляют тем, что будет происходить в базе данных при удалении строки. Аналогично правила вставки и обновления управляют тем, что будет происходить с базой данных, если строки изменяются или добавляются.

Для обеспечения ссылочной целостности внешнего ключа, когда изменяется или удаляется значение связанного первичного или уникального ключа, могут быть выполнены следующие действия:

– NO ACTION – не будет выполнено никаких действий; в клиентской программе должны быть предприняты специальные меры по поддержанию ссылочной целостности данных, иначе обновление/удаление не будет произведено и будет выдано соответствующее сообщение об ошибке;

– CASCADE – при изменении или удалении значения первичного ключа над значением внешнего ключа будут произведены те же действия. При выполнении удаления строки в главной таблице в подчинённой таблице должны быть удалены все записи, имеющие те же значения внешнего ключа, что и значение первичного (уникального) ключа удалённой строки главной таблицы. При выполнении обновления записи главной таблицы в подчинённой таблице должны быть изменены все значения внешнего ключа, имеющие те же значения, что и значение первичного (уникального) ключа изменяемой строки главной таблицы. Например, вместе с филиалом фирмы будут удаляться все агенты данного филиала.;

– SET DEFAULT – значения внешнего ключа всех соответствующих строк в подчинённой таблице устанавливаются в значение по умолчанию, заданное в предложении DEFAULT для этого столбца. Например, при удалении филиала фирмы все агенты могут быть переведены в другой филиал;

– SET NULL – значения внешнего ключа всех соответствующих строк в подчинённой таблице устанавливаются в пустое значение NULL.

```
CREATE TABLE JOB (
 JOB_CODE          JOBCODE NOT NULL,
 JOB_GRADE         JOBGRADE NOT NULL,
 JOB_COUNTRY       COUNTRYNAME,
MIN_SALARY        NUMERIC(18,2) DEFAULT 0 NOT NULL,
 MAX_SALARY        NUMERIC(18,2) NOT NULL,
 LANGUAGE_REQ      VARCHAR(15) [1:5],
 PRIMARY KEY (JOB_CODE, JOB_GRADE, JOB_COUNTRY),
 FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY(COUNTRY)
  ON UPDATE CASCADE
  ON DELETE SET NULL,
 CONSTRAINT CHK_SALARY CHECK(MIN_SALARY < MAX_SALARY)
```

или

```
ALTER TABLE MY_TABLE
  ADD CONSTRAINT FK_MY_TABLE_SOME_ID FOREIGN KEY(SOME_ID)
        REFERENCES SOME_TABLE(ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE;
```

Ограничение уникальности (UNIQUE). Уникальный ключ – дополнительная возможность ограничения значений записей таблицы с целью профилактики занесения в нее двух и более записей, имеющих одинаковое значение указанного столбца (столбцов). В отличие от PRIMARY KEY количество уникальных ключей в таблице не ограничено (точнее, ограничено максимальной комбинаторной суммой вариантов комбинации имен доменов, входящих в таблицу). Уникальный ключ также называется «альтернативным», и чаще всего предназначен не для однозначной идентификации столбца, как первичный ключ, а для указания столбца, который позволяет осуществить дополнительную идентификацию строки. Например – номер паспорта, код ИНН, номер пенсионной страховки, и т.д.

В отличие от первичного ключа, в таких полях допускаются значения NULL для любого количества строк. Таким образом, можно определить ограничение уникальности для столбцов, не имеющих ограничения NOT NULL.

```
CREATE TABLE MY_TABLE(
  ...
  FIELD1 INTEGER NOT NULL UNIQUE,
  ...
);
```

так и при описании таблицы (рекомендуемый вариант, с указанием имени CONSTRAINT)

```
CREATE TABLE MY_TABLE(
  ...
  FIELD1 INTEGER NOT NULL,
  FIELD2 INTEGER NOT NULL,
  ...
  CONSTRAINT UNQ_MY_TABLE_1 UNIQUE(FIELD1, FIELD2),
  ...
);
```

NULL в уникальных ключах. Для уникальных ключей, содержащих несколько столбцов, логика следующая:

– во всех столбцах, входящих в уникальный ключ, нет ограничения NOT NULL;

– разрешено хранение значения NULL в столбцах, входящих в уникальный ключ, в нескольких строках;

– разрешены строки, имеющие в одном из столбцов уникального ключа значение NULL, а остальные столбцы заполнены значениями и эти значения различны хотя бы в одном из них;

– запрещены строки, имеющие в одном из столбцов уникального ключа значение NULL, а остальные столбцы заполнены значениями, и эти значения имеют совпадения хотя бы в одном из них.

Примечание. Для уникальных ключей, содержащих несколько столбцов, допускаются дубликаты значений NULL только тогда, когда они помещаются во все столбцы ограничения. В остальных случаях значения NULL будут рассматриваться как одно из обычных значений (не UNKNOWN).

```
RECREATE TABLE t(x int, y int, z int, unique(x,y,z));
INSERT INTO t values(NULL, 1, 1);
INSERT INTO t values(NULL, NULL, 1);
INSERT INTO t values(NULL, NULL, NULL);
INSERT INTO t values(NULL, NULL, NULL); -- Разрешено
INSERT INTO t values(NULL, NULL, 1); -- Запрещено
```

Ограничение CHECK. Ограничение CHECK задаёт условие, которому должны удовлетворять значения, помещаемые в данный столбец. Условие – это логическое выражение, называемое также предикат, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неизвестно). Условие считается выполненным, если предикат возвращает значение TRUE или UNKNOWN (эквивалент NULL). Если предикат возвращает FALSE, то значение не будет принято. Это условие используется при добавлении в таблицу новой строки (оператор INSERT) и при изменении существующего значения столбца таблицы (оператор UPDATE), а также операторов, в которых может произойти одно из этих действий (UPDATE или INSERT, MERGE).

Примечание. При использовании предложения CHECK для столбца, базирующегося на домене, следует помнить, что выражение в CHECK лишь дополняет условие проверки, которое может уже быть определено в домене.

На уровне столбца выражение в предложении CHECK ссылается на входящее значение с помощью ключевого слова VALUE, так же как предложение CHECK в определении домена:

```
...
CURRENCY CHAR(3) NOT NULL
CONSTRAINT CHECK_CURRENCY
CHECK (VALUE IN ('AUD','EUR','GBP','RUR','USD','YEN')),
...
```

На уровне таблицы выражение в предложении CHECK ссылается на входящее значение по идентификатору столбца:

```
...
CONSTRAINT CHECK_EXCHANGE
CHECK (CURRENCY IN ('AUD','EUR','GBP','RUR','USD','YEN')
AND CURRENCY <> EXCHANGE_CURRENCY),
...
```

3.3 Порядок выполнения лабораторной работы

1. Проанализировать схему БД своего варианта задания (вариант то же, что и в лабораторной работе №1), выделить и классифицировать все существующие связи, определить необходимые ограничения целостности.
2. Создать все еще не созданные таблицы, изменить существующие таким образом, чтобы они могли участвовать в связях (описание ALTER TABLE).
3. Установить связи между таблицами.
4. Проверить работу ограничений целостности (каскадирование удаления, модификации и др.)
5. Подготовить и защитить отчет.

3.4 Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.
2. Отчет должен содержать исходные и модифицированные таблицы.
3. В отчете необходимо привести список связей по арности, описать ограничения целостности, допустимость Null-значений, идентифицируемость.
4. Отражение работы по проверке целостности данных.

3.5 Контрольные вопросы

1. Требования к ссылочной целостности данных.
2. Типы связей между отношениями.
3. Стандартная нотация ER-диаграмм.
4. Нормальные формы для баз данных.
5. Необходимость процесса нормализации БД.
6. Приведение БД к нормальной форме Бойса-Кодда.
7. Обосновать в какой нормальной форме находится полученная схема БД.
8. Способы повышения надежности данных.

ЛАБОРАТОРНАЯ РАБОТА № 4
Язык SQL. Запросы на основе нескольких таблиц

4.1 Цель работы

Изучить способы получения информации из нескольких таблиц, способы выполнения и принцип действия рекурсивных запросов и научится использовать вложенные подзапросы.

4.2 Основные положения

Запросы к нескольким таблицам являются наиболее часто используемым типом запросов. Реляционные БД нормализованы, и хранимая информация разбита по большому количеству таблиц**.**

SQL JOIN – используются для запроса данных из двух или нескольких таблиц связанных между собой ключами.

Ключом является столбец (или комбинация столбцов) с уникальным значением для каждой строки. Каждое значение первичного ключа должно быть уникальным в пределах таблицы. Цель состоит в том, чтобы связать данные всех таблиц вместе, не повторяя все данные в каждой таблице.

Есть таблица «Persons»:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Есть таблица «Orders»:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Столбец «P_Id» является первичным ключом в таблицы «Persons». Это означает, что никакие две строки не могут иметь одинаковый «P_Id».

Столбец «O_Id» является первичным ключом в таблицы «Orders» и колонка «P_Id» относится к колонке «P_Id» в таблице «Persons».

Таким образом, связь между таблицами обеспечивается с помощью колонки «P_Id».

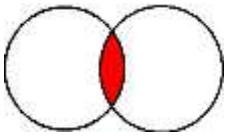
Существуют следующие типы JOIN, доступные к использованию в SQL:

- JOIN (или INNER JOIN) – возвращает строки, когда есть хотя бы одно совпадение в обеих таблицах;
- LEFT JOIN – возвращает строки из левой таблицы, даже если их нет в правой таблице;
- RIGHT JOIN – возвращает строки из правой таблицы, даже если их нет левой таблице;
- FULL JOIN – возвращает строки, когда есть хоть одно совпадение в любой из таблиц;
- CROSS JOIN – возвращает все возможные сочетания каждой строки с каждой.

INNER JOIN (естественное соединение) возвращает строки, когда есть хотя бы одно совпадение в обеих таблицах.

Синтаксис SQL INNER JOIN

```
SELECT column_name(s)
FROM table_name1
[INNER] JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```



Замечание: INNER JOIN это тоже что и JOIN.

Пример SQL INNER JOIN: Необходимо выбрать все лица, имеющие какие-либо заказы. Для этого используем такой запрос:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
INNER JOIN Orders ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

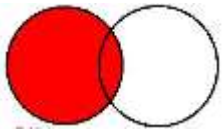
Результат запроса:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

LEFT JOIN возвращает строки из левой таблицы(table_name1), даже если их нет в правой таблице (table_name2).

Синтаксис SQL LEFT JOIN

```
SELECT column_name(s)
FROM table_name1
LEFT [OUTER] JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```



Замечание: в некоторых базах данных LEFT JOIN имеет имя LEFT OUTER JOIN.

Пример SQL LEFT JOIN. Теперь необходимо получить список всех лиц и их заказов из таблицы выше.

Для этого используем такой запрос:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
LEFT JOIN Orders ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

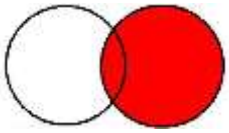
Результат запроса:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	

RIGHT JOIN возвращает строки из правой таблицы(table_name2), даже если их нет левой таблице (table_name1).

Синтаксис SQL RIGHT JOIN

```
SELECT column_name(s)
FROM table_name1
RIGHT [OUTER] JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```



Замечание: в некоторых базах данных RIGHT JOIN имеет имя RIGHT OUTER JOIN.

Пример SQL RIGHT JOIN. Теперь необходимо получить список всех лиц и их заказов из таблицы выше.

Для этого используем такой запрос:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
RIGHT JOIN Orders ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

Результат запроса:

LastName	FirstName	OrderNo
		34764
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	44678
Pettersen	Kari	77895

CROSS JOIN (декартово произведение) это объединение SQL по которым каждая строка одной таблицы объединяется с каждой строкой другой таблицы. Это объединение редко требуется.

Синтаксис SQL CROSS JOIN

```
SELECT column_name(s)
FROM table_name1
CROSS JOIN table_name2
```

Пример SQL CROSS JOIN. Необходимо вывести все возможные сочетания каждой строки из таблицы «Persons» с каждой строкой таблицы «Orders».

Для этого используем такой запрос:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
CROSS JOIN Orders
```

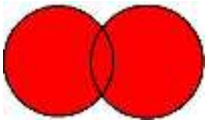
Результат запроса:

LastName	FirstName	OrderNo
Hansen	Ola	77895
Hansen	Ola	44678
Hansen	Ola	22456
Hansen	Ola	24561
Hansen	Ola	34764
Svendson	Tove	77895
Svendson	Tove	44678
Svendson	Tove	22456
Svendson	Tove	24561
Svendson	Tove	34764
Pettersen	Kari	77895
Pettersen	Kari	44678
Pettersen	Kari	22456
Pettersen	Kari	24561
Pettersen	Kari	34764

FULL JOIN возвращает строки, когда есть хоть одно совпадение в любой из таблиц.

Синтаксис SQL FULL JOIN

```
SELECT column_name(s)
FROM table_name1
FULL [OUTER] JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```



Пример SQL FULL JOIN. Необходимо получить список всех людей и заказов.

Для этого используем такой запрос:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
FULL JOIN Orders ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

Результат запроса:

LastName	FirstName	OrderNo
		34764
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	44678
Pettersen	Kari	77895
Svendson	Tove	

4.2.1 Простое соединение двух таблиц

Структура, используемых отношений (Salespeople, Customers, Orders) и их содержимое приведено в лабораторной работе №2 п. 2.2 Тестовая база данных.

Существует два основных способа соединения таблиц – с помощью оператора JOIN языка SQL, и с помощью условия в разделе WHERE. Например, соединить таблицу Customers и таблицу Salespeople по полю city можно следующим образом:

```
SELECT Customers.cname,
       Salespeople.sname,
       Salespeople.city
FROM Salespeople,
     Customers
WHERE Salespeople.city = Customers.city;
```

Для выполнения данного запроса сервер произведет декартово произведение двух таблиц, после чего выполнит операцию селекции нужных строк, затем – проекцию нужных полей. Декартово произведение – это операция, которая требует для своего выполнения максимальное количество памяти и процессорного времени, по сравнению с другими известными операциями. Данный способ неэффективен.

В случае если используется, оператор JOIN, запрос будет выглядеть так:

```
SELECT Customers.cname,
       Salespeople.sname,
       Salespeople.city
FROM Salespeople
     JOIN Customers
     ON Salespeople.city = Customers.city;
```

В данном случае сервер БД не будет производить декартово произведение, сервер воспользуется индексами для определения совпадающих значений поля city, выберет строки с совпадающими значениями, а затем произведет проекцию. Данная операция происходит на порядок быстрее. В случае, если по полям учавствующим в соединении, не создан индекс, сервер вынужден производить декартово произведение.

Можно сделать следующие выводы:

- если соединение таблиц происходит по ключам, выгоднее использовать JOIN;
- если запрос выполняется медленно, нужно создать индекс по полю – параметру соединения (CREATE INDEX).

4.2.2 Соединение более чем двух таблиц

Соединение более чем двух таблиц по форме не отличается от простого соединения. Например, чтобы соединить три таблицы по некоторому условию, можно записать:

```
SELECT onum, cname, Orders.cnum, Orders.snum
FROM Salespeople, Customers, Orders
WHERE Customers.city <> Salespeople.city
      AND Orders.cnum = Customers.cnum
      AND Orders.snum = Salespeople.snum;
```

Тот же запрос, переписанный с использованием JOIN, выглядит более сложно:

```
SELECT onum, cname, Orders.cnum, Orders.snum
FROM   (Orders   JOIN   Customers   ON   Orders.cnum   =   Customers.cnum)
      JOIN Salespeople ON Orders.snum = Salespeople.snum
WHERE Customers.city <> Salespeople.city;
```

4.2.3 Псевдонимы и рекурсивные объединения

На практике часто встречается ситуация, когда необходимо соединять таблицу саму с собой. Запрос, выполняющий такое соединение, называется рекурсивным.

Таблица «Студент» (Students) содержит следующие атрибуты:

- CNAME – имя студента;
- RATING – рейтинг студента;
- SPES – специальность.

Таблица «Студент»

CNAME	RATING	SPES
Иванов	42	ИС
Калинин	46	А
Петров	42	А
Сидоров	34	ИС
Шахов	46	М

Например, если необходимо вывести все пары студентов с одинаковым рейтингом, можно записать следующий запрос:

```
SELECT first.cname, second.cname, first.rating
FROM Students first, Students second
WHERE first.rating = second.rating;
```

Здесь в списке вывода SELECT указываются поля cname и rating таблицы first, и поле cname таблицы second. В разделе FROM указывается, что first и second – просто псевдонимы для таблицы Students. Для выполнения запроса сервер создаст две копии таблицы Students, одну с именем first, другую с именем second, выполнит запрос так, как будто это две разные таблицы, и уничтожит копии. Естествнно, сервер физически не создает копий таблиц, но с псевдонимами в запросе он работает так, будто это две разные таблицы.

Можно заметить, что вывод запроса будет повторять каждую пару дважды - сначала «Иванов - Петров», затем «Петров - Иванов». Кроме того, вывод содержит строки «Иванов - Иванов», «Петров - Петров». Это происходит потому, что сервер берет первую строку из первого псевдонима и сравнивает ее со всеми строками из второго псевдонима. Будут выбраны строки «Иванов - Иванов» и «Иванов - Петров». Затем он переходит к следующей строке и снова сравнивает ее со всеми строками из второго псевдонима, и так далее. Будут выбраны строки «Петров - Иванов» и «Петров - Петров».

Чтобы избежать дубликатов, надо определить еще одно условие, налагающее отношение порядка на сравниваемые строки. Например, сравнивать имена.

```
SELECT first.cname, second.cname, first.rating
FROM Students first, Students second
WHERE first.rating = second.rating AND
      first.cname < second.cname;
```

4.2.4 Вложенные подзапросы

Вложенные подзапросы так же служат для получения информации из нескольких таблиц. С их помощью можно выполнять рекурсивные запросы. Очень часто на практике встречаются ситуации, когда запрос выражается очень сложно через соединения, и легко – через вложенный подзапрос, и наоборот. На конкретном сервере БД запрос с использованием JOIN может выполняться очень долго, а с использованием подзапроса – быстро.

Пример запроса с вложенным подзапросом:

```
SELECT * FROM Orders
WHERE snum = (SELECT snum FROM Salespeople
              WHERE sname = 'Motika');
```

Так как подзапрос стоит после знака равенства, он должен возвращать только одно значение. В случае если подзапрос вернет более чем одно значение, произойдет ошибка.

Внимание! При записи подзапроса допустима следующая форма:

<имя/константа> <оператор> <подзапрос>,

а не <подзапрос> <оператор> <имя/константа> или

< подзапрос > < оператор > < подзапрос >.

Во вложенных подзапросах можно использовать агрегатные функции:

```
SELECT * FROM Orders
WHERE amt >
  (SELECT AVG(amt) FROM Orders
   WHERE odate = 10/04/1990 );
```

Ограничение на вложенный подзапрос то же самое – он должен возвращать единственное значение.

В случае, если подзапрос возвращает несколько записей, вместо операций сравнения нужно использовать IN:

```
SELECT * FROM Orders
WHERE snum IN
  (SELECT snum FROM Salespeople
   WHERE city = "LONDON");
```

Данный запрос более просто записывается с использованием соединения:

```
SELECT onum, amt, odate, cnum, Orders.snum
FROM Orders, Salespeople
WHERE Orders.snum = Salespeople.snum
  AND Salespeople.city = "London";
```

Допустимо использовать выражение, основанное на столбце, а не просто сам столбец в предложении SELECT подзапроса:

```
SELECT * FROM Customers
WHERE cnum =
  (SELECT snum + 1000 FROM Salespeople
   WHERE sname = Serres);
```

Также допустимы подзапросы в выражении HAVING:

```
SELECT rating, COUNT(DISTINCT cnum) FROM Customers
GROUP BY rating
HAVING rating >(SELECT AVG(rating) FROM Customers
                WHERE city = 'San Jose');
```

4.2.5 Индексы

Индексы позволяют быстрее находить нужные данные, без чтения всей таблицы данных. Пользователь не может увидеть индексы, они просто используются для ускорения поиска/запроса. При добавлении ограничений первичного ключа, внешнего ключа или ограничения уникальности будет неявно создан одноименный индекс. Так, например, при выполнении следующего оператора будет неявно создан индекс PK_COUNTRY.

```
ALTER TABLE COUNTRY ADD CONSTRAINT PK_COUNTRY PRIMARY KEY(ID);
```

Замечание: обновления таблицы с индексами занимает большее время, чем обновления таблицы без индексов (потому что индексы тоже нуждаются в обновлении). Итак, рекомендуется создавать только один индекс на таблицу (одна колонка) в которой будет часто производится поиск.

CREATE INDEX – данный запрос используется для создания индексов в таблице.

Синтаксис SQL CREATE INDEX. Повторяющиеся значения допускаются:

```
CREATE INDEX index_name
ON table_name (column_name)
```

Синтаксис SQL CREATE UNIQUE INDEX. Повторяющиеся значения не допускаются:

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

Замечание: синтаксис для создания индексов не одинаковый в базах данных. Поэтому проверьте синтаксис перед созданием индексов в базе данных.

Пример CREATE INDEX. SQL запрос создает индекс с именем «PIndex» в колонке «LastName» таблице «Persons»:

```
CREATE INDEX PIndex
ON Persons (LastName)
```

Если вы хотите создать комбинированный индекс, то можно перечислить имена столбцов в скобках через запятую:

```
CREATE INDEX PIndex
ON Persons (LastName, FirstName)
```

4.3 Порядок выполнения лабораторной работы

1. Записать запросы, соединяющие две таблицы с помощью JOIN и без него.
2. Записать запросы, соединяющие более чем две таблицы с помощью JOIN и без него.
3. Продемонстрировать следующие возможности SQL:
   - использование псевдонимов на примере рекурсивного запроса.
   - привести пример запроса с подзапросом
   - использование агрегатных функций в подзапросе
   - подзапросы, возвращающие единственное и множественные значения
   - подзапросы, использующие вычисление
   - использование подзапросов в HAVING
4. Написать отчет.

4.4 Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.

2. Отчет должен содержать исходные данные, тексты запросов и результаты их выполнения.

4.5 Контрольные вопросы

1. Операции реляционной алгебры (операторной формы записи).
2. Бинарные операторы над отношениями, операторная форма записи.
3. В чем различие соединения таблиц по условию и с использованием JOIN?
4. Свойства операции соединения.
5. В чем различие вложенных запросов и запросов с соединением?
6. Какие формы записи подзапроса недопустимы?
7. В чем особенность подзапроса, перед которым стоит знак арифметического сравнения?

ЛАБОРАТОРНАЯ РАБОТА № 5
Язык SQL. Коррелированные вложенные подзапросы

5.1 Цель работы

Ознакомится с принципом работы коррелированных подзапросов

5.2 Основные положения

5.2.1 Коррелированные вложенные подзапросы

Вложенные подзапросы и операция JOIN предоставляют большие возможности по манипулированию данными из нескольких таблиц. Существует еще один способ - использовать коррелированные подзапросы, он более сложен для понимания, но в некоторых случаях позволяет формулировать запросы самым коротким образом. Кроме того, есть некоторые вещи, доступные для соединения и недоступные для вложенных подзапросов, и наоборот.

Например, при использовании подзапроса становятся возможным использовать агрегатные функции в предикате запроса. При использовании соединений в выводе запроса могут участвовать поля из всех таблиц соединения.

Коррелированным называется подзапрос, который использует псевдонимтаблицы определенный не во вложенном запросе, а во внешнем. При этом вложенный запрос выполняется много раз - для каждой строки внешней таблицы. Пусть требуется найти всех покупателей, совершивших покупки 10.03.1990. Можно записать следующий коррелированный запрос:

```
SELECT *
FROM Customers outer
WHERE 10.03.1990 IN
        (SELECT odate
         FROM Orders inner
         WHERE outer.cnum = inner.cnum);
```

Логика работы запроса такова: внутренний запрос производит соединение таблицы покупателей и таблицы покупок (outer.cnum = inner.cnum). Для тех строк, у которых условие WHERE выполняется, запрос возвращает дату покупки. Таким образом, для каждого покупателя формируется множество дней, когда он делал покупки. Полученное множество передается во внешний запрос. В случае если 10.03.1990 входит в сформированное множество, внешний запрос возвращает всю строку (*) строку из таблицы Customers.

Приведем процесс выполнения коррелированного подзапроса в алгоритмической форме:

1. Выбрать строку из таблицы во внешнем запросе (строка-кандидат).
2. Сохранить значения полей строки-кандидата в псевдониме.
3. Выполнить подзапрос. Везде, где найдена ссылка на псевдоним из внешнего запроса, подставлять значения из текущей строки-кандидата. Использование значения из строки-кандидата называется внешней ссылкой.
4. Оценить предикат внешнего запроса на основе результатов подзапроса. Если предикат верен - строка выбирается для вывода
5. Перейти на следующую строку во внешнем запросе.

Подобный запрос можно гораздо легче записать другим способом. Например:

```
SELECT DISTINCT *
FROM Customers, Orders
WHERE Customers.cnum = Orders.cnum
       AND Orders.odate = 10.03.1990;
```

Более показательным является следующий пример. Пусть необходимо вывести имена и номера всех продавцов, которые имеют более одного заказчика

```
SELECT snum, sname
FROM Salespeople main
WHERE 1 <
 (SELECT COUNT(*)
  FROM Customers
  WHERE snum = main.snum);
```

В случае, если перед именем поля не указывается имя псевдонима, то данное поле относится к текущему подзапросу. Таким образом, поле snum во вложенном подзапросе относится к таблице Customers, а не к Salespeople. В случае, если такого поля в соответствующей таблице нет, оно ищется в подзапросе верхнего уровня. Так, если бы поле snum отсутствовало в таблице Customers, оно относилось бы к Salespeople.

5.2.2 Соотнесение таблицы со своей копией

Коррелированные запросы можно писать, основываясь только на одной таблице. Данное свойство делает их незаменимыми при написании аналитических запросов (т.е. запросов, производящих сложный анализ данных). Например, пусть нам необходимо найти все покупки со значениями суммы покупки выше среднего значения для каждого покупателя (т.е. надо найти все покупки данного покупателя, найти среднюю сумму покупок, и выдать все покупки со стоимостью выше средней):

```
SELECT *
FROM Orders outer
WHERE amt > =
 (SELECT AVG(amt)
  FROM Orders inner
  WHERE inner.cnum = outer.cnum );
```

Существуют так называемые специальные операторы SQL, они имеют смысл только для подзапросов. Отличительная особенность специальных операторов - они принимают подзапрос как аргумент, точно так же, как это делает IN.

5.2.3 Оператор EXISTS

EXISTS(X) – булевский оператор. Он получит значение "истинна", если запрос X вернет хоть одну строку. Допустим, нам необходимо узнать список продавцов, у которых есть более одного покупателя

```
SELECT DISTINCT snum
FROM Customers outer
WHERE EXISTS
 (SELECT *
  FROM Customers inner
  WHERE inner.snum = outer.snum
  AND inner.cnum < > outer.cnum );
```

Для каждой строки-кандидата из внешнего запроса (продавец, проверяемый в настоящее время), внутренний запрос находит строки, у которых совпадают значения поля snum (номер продавца), но не совпадают значение поля cnum (номер покупателя). Если не указать DISTINCT, каждый продавец будет выбран один раз для каждого своего заказчика.

На практике очень часто приходится использовать EXISTS вместе с NOT. Допустим, нам необходимо вывести список продавцов, за которыми числится только один покупатель

```
SELECT DISTINCT snum
FROM Customers outer
WHERE NOT EXISTS
 (SELECT *
  FROM Customers inner
  WHERE inner.snum = outer.snum
  AND inner.cnum < > outer.cnum );
```

Кроме EXISTS в подзапросах возможно использование еще двух операторов – ANY и ALL.

5.2.4 Оператор ANY

Оператор ANY становится верным, если значение из верхнего подзапроса совпадает, по крайней мере, с одним значением из вложенного подзапроса. Например, если значение - кандидат равно 1, а вложенный подзапрос вернул {1, 2, 3}, оператор ANY станет верным, (внешний запрос проверяет на равенство). Например, если нам нужно найти всех продавцов, которые живут в тех же городах, что и покупатели, можно записать следующий запрос:

```
SELECT *
FROM Salespeople
WHERE city = ANY
 (SELECT city
  FROM Customers );
```

Существует синоним оператора ANY – SOME. Так как SQL похож на английский, то одни предложения, верно, звучат с ANY, другие – с SOME. Действие операторов эквивалентно.

5.2.5 Оператор ALL

Вторым допустимым оператором является ALL. Действие его противоположно оператору ANY. Оператор ALL становится верным, если все значения из вложенного подзапроса равны значению-кандидату из внешнего запроса. Например, если значение - кандидат равно 1, а вложенный подзапрос вернул {1, 1, 1}, оператор ALL станет верным, (внешний запрос проверяет на равенство).

Например, если нам необходимо найти всех продавцов, у которых рейтинг выше, чем у любого продавца из Рима, можно записать следующий запрос:

```
SELECT *
FROM Customers
WHERE rating > ALL
 (SELECT rating
  FROM Customers
  WHERE city = Rome);
```

Операторы ANY и ALL можно выразить через EXISTS в коррелированном подзапросе, в явном виде они нужны лишь для упрощения записи запроса. Обратное утверждение не верно – т.е. не все то, что можно выполнить с помощью EXISTS, можно сделать с помощью ANY и ALL.

Замечание 1: Когда говорят, что значение больше (или меньше) чем любое (ANY) из набора значений, это то же самое, что сказать, что оно больше (или меньше) чем любое отдельно взятое из этих значений. И наоборот, сказать, что некоторое значение не равно всему (ALL) набору значений, это то же самое, что сказать, что в наборе нет такого же значения.

Замечание 2: В случае, если подчиненный запрос вернул пустое множество, оператор ALL становится верным, а ANY – ложным.

5.3 Порядок выполнения лабораторной работы

1. Записать запрос, соединяющий таблицу со своей копией.
2. Привести пример коррелированного запроса, использующего две разные таблицы.
3. Продемонстрировать следующие возможности SQL
   - работу оператора EXISTS;
   - работу оператора ALL;
   - работу оператора ANY.
4. Написать отчет.

5.4 Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.

2. Отчет должен содержать исходные данные, тексты запросов и результаты их выполнения.

5.5 Контрольные вопросы

1. Коррелированные вложенные подзапросы.
2. Для чего таблицам назначается псевдоним? Приведите примеры назначения и использования псевдонимов в запросах.
3. Операторы: EXISTS, ALL, ANY – назначение, правила использования.
4. Какой из операторов EXISTS, ALL, ANY являются альтернативой друг другу? Приведите примеры.

ЛАБОРАТОРНАЯ РАБОТА № 6
Пользовательские представления

6.1 Цель работы

Ознакомится с принципом работы пользовательских представлений, продемонстрировать работу на примере.

6.2 Основные положения

6.2.1 Пользовательские представления

Информация в реляционной базе данных разбита по множеству таблиц таким образом, чтобы обеспечить безизбыточное хранение информации. Подобный принцип хорош для правильного функционирования базы данных, но работать пользователю с таблицами БД напрямую неудобно.

Например, расписание движения поездов на железной дороге может быть представлено таблицей, в соответствии с рисунком 6.1.

Номер поезда	Станция Отправления	Станция Прибытия	Время отправления	Время прибытия
012	Москва	Петушки	12:30	14:20

Рисунок 6.1 – Таблица «Расписание»

В реляционной БД можно предложить следующий набор таблиц (представленных на рисунках 6.2-6.4) для хранения информации о расписании.

Номер станции	Наименование станции
1	Москва
2	Петушки

Рисунок 6.2 – Таблица «Станция»

Номер поезда	Номер станции отправления	Номер станции прибытия
012	1	2

Рисунок 6.3 – Таблица «Поезд»

Номер поезда	Время отправления	Время прибытия
012	12:30	14:20

Рисунок 6.4 – Таблица «Расписание»

Очевидно, что если просто вывести содержимое таблицы «Расписание», оно мало, чем поможет пассажиру, который не помнит маршруты поездов по номерам. Для того чтобы выводить данные из БД в удобном для пользователя виде, существуют пользовательские представления (VIEW).

Пользовательское представление – это объект базы данных, представляющий из себя запрос на выборку, каким-либо образом комбинирующий информацию из базовых таблиц базы данных. Часто о представлениях говорят, как о «виртуальных» таблицах. Физически данные из исходных таблиц в представления не переносятся, но со стороны представления выглядят как полноправные таблицы.

Так как представления - это объекты БД, то они сохраняются в файле БД. Представления можно определять на одной таблице, на наборе таблиц, в том числе можно использовать и другие представления. В базе данных хранится только определение пользовательского

представления. Когда представление запрашивается клиентской программой, выполняется запрос на выборку из представления. Если данные из базовых таблиц представления изменены другим пользователем, то представление автоматически перестраивается сервером БД для того, чтобы отобразить изменения.

Представление может быть составлено из:

- подмножества столбцов одной таблицы (выводятся не все столбцы таблицы);
- подмножества строк одной таблицы (выводятся не все строки таблицы);
- подмножества строк и столбцов одной таблицы;
- подмножества строк и столбцов нескольких таблиц (объединения).

Преимущества представлений:

- упрощенный доступ к данным;
- персонализированный доступ к данным. На одном и том же наборе таблиц можно сделать несколько представлений. Например, одно будет показывать информацию необходимую клерку, а другое – менеджеру;
- независимость от изменений базовых таблиц БД. Можно поменять структуру БД, но переписать представление таким образом, чтобы оно отображало те же данные;
- защита данных. В представлении можно выводить только те данные, которые должны быть видимы пользователю. Например, можно сделать список сотрудников сделать доступным для открытого просмотра, но не выводить поле «Зарплата» в нем.

Создание представлений

Представления создаются оператором CREATE VIEW. Синтаксис оператора:

```
CREATE VIEW name [(view_col [, view_col :])]
AS < select > [WITH CHECK OPTION];
```

где select – запрос на выборку. Представление нельзя создавать как select из хранимых процедур.

view_col – имена столбцов, выводимых представлением.

Если имена столбцов не указать, то будут использованы имена столбцов из запроса. Если имена столбцов указаны, они должны следовать в том же порядке и в таком же количестве, как возвращаются запросом на выборку. Имя столбца должно быть уникальным среди имен столбцов данного представления. Если в **базовом запросе присутствуют выражения**, имена столбцов должны быть **указаны обязательно**.

WITH CHECK OPTION - если указана данная опция, то запрещаются операции INSERT и UPDATE на обновляемых представлениях, если они не удовлетворяют условию WHERE в формулировке представления. То есть позволяются только те операции, которые будут «видимы» представлением.

Обновляемые и не обновляемые представления

Представление является обновляемым, и может распространять изменения, вносимые в представление на базовые таблицы, только при выполнении некоторых условий. Если условия не выполняются, то представление может служить только для чтения. Термины «обновляемое» и «только для чтения» относятся к содержимому представления, а не к его объявлению.

Условия обновляемости представлений:

- представление является подмножеством одной таблицы, либо другого обновляемого представления;
- все столбцы базовых таблиц не включенные в представление должны позволять состояние NULL;
- оператор SELECT не должен содержать подзапросов, слова DISTINCT, раздела HAVING, агрегатных функций, объединения таблиц, пользовательских функций, хранимых процедур.

Примеры представлений

Описание базовых отношений приведено в Лабораторной работе №1. Пример редактируемого представления:

```
CREATE VIEW EMP_MNGRS (FIRST, LAST, SALARY) AS
   SELECT FIRST_NAME, LAST_NAME, SALARY
   FROM EMPLOYEE
   WHERE JOB_CODE = 'Mngr';
```

В следующем представлении используется подзапрос, поэтому представление не редактируемое:

```
CREATE VIEW ALL_MNGRS AS
  SELECT FIRST_NAME, LAST_NAME, JOB_COUNTRY
  FROM EMPLOYEE
  WHERE JOB_COUNTRY IN
        (SELECT JOB_COUNTRY
         FROM JOB
         WHERE JOB_TITLE = 'manager');
```

Не редактируемое представление, использующее соединение таблиц:

```
CREATE VIEW PHONE_LIST AS
  SELECT EMP_NO, FIRST_NAME, LAST_NAME,
         PHONE_EXT, LOCATION, PHONE_NO
  FROM EMPLOYEE, DEPARTMENT
  WHERE EMPLOYEE.DEPT_NO=DEPARTMENT.DEPT_NO
```

Удаление представлений

Удалить представление можно с помощью команды DROP VIEW:

```
DROP VIEW name;
```

Здесь name - имя удаляемого представления. Не существует способов изменить существующее представление. Можно удалить старое представление и заново создать новое.

6.3 Порядок выполнения лабораторной работы

1. В соответствии с вариантом задания создать представления.
2. Просмотреть хранимые данные, используя представления.
3. Используя редактируемое преставление внести изменения в таблицу. Просмотреть полученный результат.
4. Удалить полученные преставления.
5. Написать отчет.

6.4 Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.
2. Отчет должен содержать:
   - исходные данные;
   - описание представлений;
   - тексты запросов и результаты их выполнения;
   - объяснение полученных результатов.
3. Задание на работу (см. приложение Г).

6.5 Контрольные вопросы

1. Что такое «пользовательское представление»?
2. Перечислить условия редактируемости представлений.
3. Назначение пользовательских представлений.
4. Привести примеры пользовательских представлений.
5. Отличие представлений от таблиц.
6. Организация ограничения доступа пользователей к набору данных.
7. Условие обеспечения целостности данных.
8. Внешняя модель и создание представлений при разработке БД.

ЛАБОРАТОРНАЯ РАБОТА № 7
Манипулирование базой данных. Реляционная алгебра и язык SQL

7.1 Цель работы

Изучить основы реляционной алгебры, как базового средства манипулирования. Выработать у обучающихся практические навыки по работе с реляционными базами данных и представлению запросов как на языке реляционной алгебре, так и SQL.

7.2 Данные, на примере которых будут показаны операции

Рейс

№ рейса	Пункт отправления	Пункт назначения	Время вылета	Стоимость
35	Севастополь	Москва	9-40	50
47	Севастополь	Санкт-Петербург	12-00	70
112	Москва	Санкт-Петербург	16-00	30

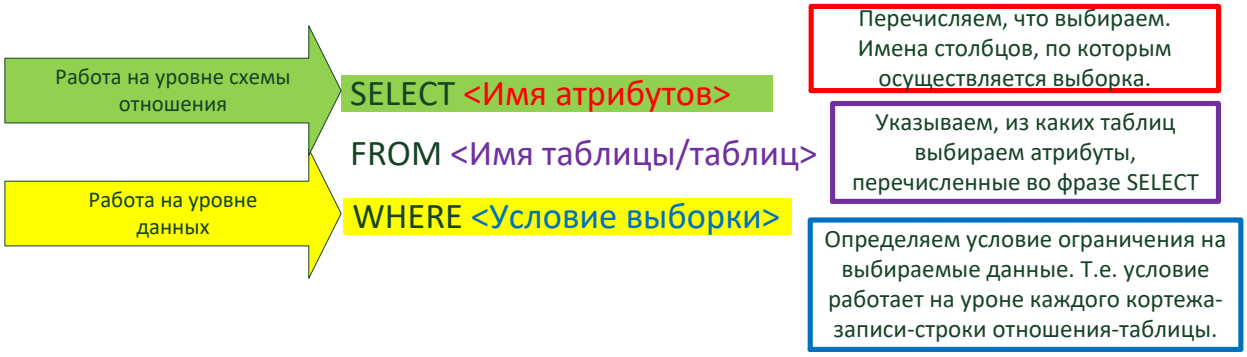
Полет

Дата	№ рейса	Код экипажа	Свободные места	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
1.10	112	15	7	Боинг-7	5
2.10	47	18	4	Боинг-7	4
3.10	35	17	3	Ту-154	2

Самолет

Тип самолета	Число экипажа	Кол-во мест	Вес груза
Ту-154	5	80	5
Боинг-7	8	110	7

7.3 Простая форма выборки

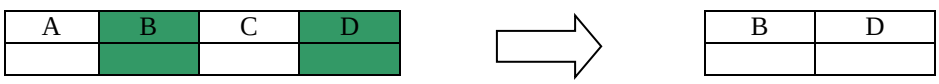


7.4 Реляционные операции

- операции над одним отношением – унарные: проекция, селекция (выборка);
- операции работы с множествами: объединение, пересечение, вычитание;
- декартово произведение и операция соединения;
- операция деления.

7.4.1 Проекция

Проекция – выборка из отношения данных по заданным атрибутам.



Запрос: Определить соотношение между типом самолета и допустимой массой перевозимого груза.

SQL:

```
SELECT тип_самолета, вес_груза
FROM Самолет
```

PA:

$R = \pi_{\text{тип\_самолета, вес\_груза}}(\text{Самолет})$

Указываем атрибуты, по которым производим проекцию на отношение

Самолет

Тип самолета	Число экипажа	Кол-во мест	Вес груза
Ту-154	5	80	5
Боинг-7	8	110	7

Результат

Тип самолета	Вес груза
Ту-154	5
Боинг-7	7

7.4.2 Селекция

Выборка-селекция – выборка из отношения кортежей с ограничением на данные.

A	B	C	D
1			aaa
5			
6			
14			aaa

A	B	C	D
1			aaa
14			aaa

Запрос: Вывести полную информацию о полете, которую осуществляет экипаж с кодом 17.

SQL:

```
SELECT *
FROM Полет
WHERE код_экипажа = 17
```

PA:

$R = \sigma_{\text{код\_экипажа} = 17}(\text{Полет})$

Условие ограничения на выбираемые кортежи

Полет

Дата	№ рейса	Код экипажа	Свободные места	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
1.10	112	15	7	Боинг-7	5
2.10	47	18	4	Боинг-7	4
3.10	35	17	3	Ту-154	2

Результат:

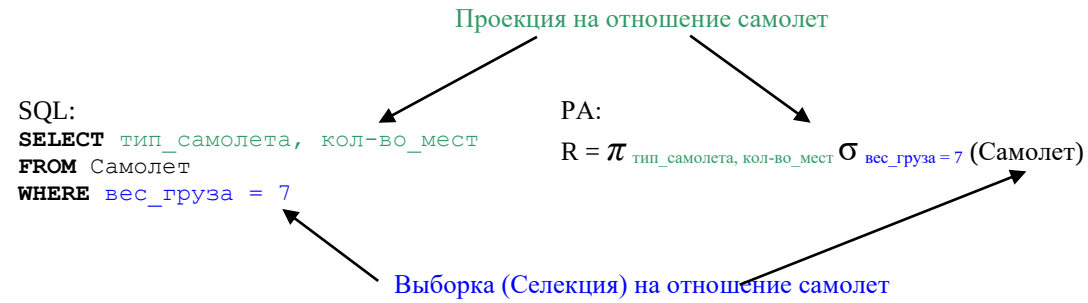
Дата	№ рейса	Код экипажа	Свободные места	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
3.10	35	17	3	Ту-154	2

7.4.3 Совмещение операций Проекция и Селекция

A	B	C	D
1	a	b	10
2	c	b	12
3	d	m	3

B	C
c	b

Запрос: Вывести информацию о типе самолета и количестве мест для самолетов с допустимым весом груза 7 т.



Самолет

Тип самолета	Число экипажа	Кол-во мест	Вес груза
Ту-154	5	80	5
Боинг-7	8	110	7

Результат:

Тип самолета	Кол-во мест
Боинг-7	110

7.4.4 Примеры решения задач на РА и SQL

1. Определить число свободных мест по всем рейсам на 20.06.18.

R = π №рейса, свободные_места (σ дата = 20.06.18 (Полет));

```
SELECT №рейса, свободные места
FROM Полет WHERE дата = "20.06.18";
```

2. Определить рейсы и время вылета из Симферополя в Москву.

R = π №рейса, время_вылета (σ пункт_отправления = “Симферополь” ∧ пункт_назначения = «Москва» (Рейс));

```
SELECT №рейса, время вылета FROM Рейс
WHERE пункт_отправления = "Симферополь" AND пункт_назначения = "Москва";
```

3. Определить типы самолетов, число членов экипажа на которых 13.

R = π тип_самолета (σ число_экипажа =13(Самолет))

```
SELECT тип_самолета FROM Самолет WHERE число_экипажа = 13;
```

7.4.5 Объединение

Операция допустима только над совместными множествами-доменами (одной природы и структуры). Т.е. операция допустима для отношений, имеющих одинаковые схемы. Если отношение имеют разные схемы, но в схемах присутствуют атрибуты, реализованные на одном домене, то операция Объединение допустима для таких отношений только при приведении этих отношений операцией проекцией (над каждым) к одинаковым схемам.

Отношение 1:

A	B	C	D

Отношение 2:

K	C	N

Шаг1. Проекция на отношение 1 по атрибуту С. Проекция на отношение 2 по атрибуту С.

C

C

Шаг 2. Объединение. Отношения имеют одинаковые схемы, операция объединения допустима.

C

Запрос: Определить тип самолета, для которых либо число членов экипажа равно 5, либо код экипажа равен 15.

Проекция: 1. приведение отношения Полет к новой схеме; 2. приведения отношения Самолет к новой схеме.

SQL:

```
SELECT тип_самолета
FROM Полет
WHERE код_экипажа = 15
UNION
SELECT тип_самолета
FROM Самолет
WHERE число_экипажа = 5
```

PA:

$R1 = \pi_{\text{тип\_самолета}} \sigma_{\text{код\_экипажа} = 15}$ (Полет)

$R2 = \pi_{\text{тип\_самолета}} \sigma_{\text{число\_экип} = 7}$ (Самолет)

$R = R1 \cup R2$

Ограничения на выборку по кортежам

UNION – признак операции объединения для SQL.

Полет

Дата	№ рейса	Код экипажа	Свободные места	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
1.10	112	15	7	Боинг-7	5
2.10	47	18	4	Боинг-7	4
3.10	35	17	3	Ту-154	2

Самолет

Тип самолета	Число экипажа	Кол-во мест	Вес груза
Ту-154	5	80	5
Боинг-7	8	110	7

Промежуточный результат:

Самолет:

Тип самолета
Ту-154

Полет:

Тип самолета
Боинг-7

Самолет ∪ Полет

Тип самолета
Ту-154
Боинг-7

7.4.6 Пересечение

Операция допустима только над совместными множествами-доменами (одной природы и структуры). Т.е. операция допустима для отношений, имеющих одинаковые схемы. Если отношение имеют разные схемы, но в схемах присутствуют атрибуты, реализованные на одном домене, то операция Пересечение допустимо для таких отношений только при приведении этих отношений операцией проекцией (над каждым) к одинаковым схемам.

Отношение 1:

A	B	C	D

Отношение 2:

K	C	N

Шаг1. Проекция на отношение 1 по атрибуту С. Проекция на отношение 2 по атрибуту С.

C

C

Шаг 2. Пересечение. Отношения имеют одинаковые схемы, операция пересечения допустима.

C

Элементы множеств, принадлежащих одновременно отношению 1 и отношению 2.

Запрос: Вывести полную информацию о полетах для самолетов с максимально допустимым весом груза до 6 т.

Неявное приведение отношения Полет к новой схеме по заданному атрибуту.

Проекция: приведение отношения Самолет к новой схеме.

SQL:

```
SELECT *
FROM Полет
WHERE тип_самолета IN
 (SELECT тип_самолета
FROM Самолет
WHERE вес_груза <= 6 )
```

PA:

$R1 = \pi_{\text{тип\_самолета}}$ (Полет)

$R2 = \pi_{\text{тип\_самолета}} \sigma_{\text{вес\_груза} <= 6}$ (Самолет)

$R = R1 \cap R2$

Итоговое отношение:

$R = \sigma_{\text{тип\_самолета}}$ (Полет)

Выборка (Селекция) кортежа: вес груза до 6.

IN – признак операции пересечения.

Полет

Дата	№ рейса	Код экипажа	Свободные места	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
1.10	112	15	7	Боинг-7	5
2.10	47	18	4	Боинг-7	4
3.10	35	17	3	Ту-154	2

Самолет

Тип самолета	Число экипажа	Кол-во мест	Вес груза
Ту-154	5	80	5
Боинг-7	8	110	7

Промежуточный результат:

Самолет:

Тип самолета
Ту-154

Полет:

Тип самолета
Ту-154
Боинг-7
Боинг-7
Ту-154

Самолет ∩ Полет

Тип самолета
Ту-154

Результат действия запроса $R = \sigma_{\text{тип\_самолета} \in R}$ (Полет):

Дата	№ рейса	Код экипажа	Свободные мест	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
3.10	35	17	3	Ту-154	2

7.4.7 Вычитание

Операция допустима только над совместными множествами-доменами (одной природы и структуры). Т.е. операция допустима для отношений, имеющих одинаковые схемы. Если отношение имеют разные схемы, но в схемах присутствуют атрибуты, реализованные на одном домене, то операция Вычитания допустима для таких отношений только при приведении этих отношений операцией проекцией (над каждым) к одинаковым схемам.

Отношение 1:

A	B	C	D

Отношение 2:

K	C	N

Шаг1. Проекция на отношение 1 по атрибуту С. Проекция на отношение 2 по атрибуту С.

C

C

Шаг 2. Вычитание. Отношения имеют одинаковые схемы, операция вычитания допустима.

C

Элементы множеств, принадлежащие отношению 1, и не принадлежащие отношению 2.

Запрос: Определить номера рейсов, которые не производились с даты A по дату B.

Неявное приведение отношений к новой схеме по заданному атрибуту.

SQL:

```
SELECT №_рейса
FROM Полет
WHERE №_рейса NOT EXISTS
 (SELECT №_рейса
FROM Полет
WHERE дата between A and B )
```

Выборка (Селекция) кортежа: дата с A по B.

PA:

R1 = $\pi_{№\_рейса}$ (Полет)

R2 = $\pi_{№\_рейса}\ \sigma_{дата > A \wedge дата < B}$ (Полет)

R =R1 – R2

NOT EXISTS– признак операции вычитания.

Полет

Дата	№ рейса	Код экипажа	Свободные места	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
1.10	112	15	7	Боинг-7	5
2.10	47	18	4	Боинг-7	4
3.10	35	17	3	Ту-154	2

Промежуточный результат:

Полет:

№ рейса
35
112
47
35

Полет (ограничения по дате с 1.10 по 2.10):

№ рейса
35
112
47

⇒

Результат действия запроса: Полет – Полет

№ рейса
∅ (*null*)

7.4.8 Декартово произведение

Единицей, над которой осуществляются действия при декартовом произведении, является кортеж отношения.

Пусть n_1 – количество атрибутов схемы отношения 1;

n_2 – количество атрибутов схемы отношения 2;

m_1 – количество кортежей отношения 1;

m_2 – количество кортежей отношения 2.

Для примера, соответственно:

$n_1 = 4$; $n_2 = 3$; $m_1 = 3$; $m_2 = 4$.

Отношение 1:

A	B	C	D
1			
2			
3			

Отношение 2:

K	C	N
1		
2		
3		
4		

отношение1 × отношение2

A	B	C	D	K	C	N
1	…			1		
…	…	…	…	…	…	…
1	…			4		
…	…	…	…	…	…	…
3	…			1		
…	…	…	…	…	…	…
3	…			4		

$m = m_1 + m_2$

$n = n_1 + n_2$

Характеристика n – мощность отношения. $n = n_1 + n_2 = 7$.

Характеристика m – арность отношения $m = m_1 + m_2 = 12$.

Запрос: Определить все возможные комбинации между существующими рейсами и типами самолетов.

SQL:

```
SELECT *
FROM Самолет, Рейс
```

PA:

R = Самолет × Рейс

Комбинация признаков декартова произведения в SQL-запросе:

- выбираются все атрибуты – участвуют полностью схемы отношений;
- во фразе FROM дано перечисление таблиц;
- отсутствует условие ограничения.

Рейс

№ рейса	Пункт отправления	Пункт назначения	Время вылета	Стоимость
35	Севастополь	Москва	9-40	50
47	Севастополь	Санкт-Петербург	12-00	70
112	Москва	Санкт-Петербург	16-00	30

Самолет

Тип самолета	Число экипажа	Кол-во мест	Вес груза
Ту-154	5	80	5
Боинг-7	8	110	7

Результат:

№ рейса	Пункт отправления	Пункт назначения	Время вылета	Стоимость	Тип самолета	Число экипажа	Кол-во мест	Вес груза
35	Севастополь	Москва	9-40	50	Ту-154	5	80	5
35	Севастополь	Москва	9-40	50	Боинг-7	8	110	7
47	Севастополь	Санкт-Петербург	12-00	70	Ту-154	5	80	5
47	Севастополь	Санкт-Петербург	12-00	70	Боинг-7	8	110	7
112	Москва	Санкт-Петербург	16-00	30	Ту-154	5	80	5
112	Москва	Санкт-Петербург	16-00	30	Боинг-7	8	110	7

7.4.9 Соединение

Операция соединения допустима только, если отношения содержат атрибуты, реализованные на одном домене.

Реализацию операции соединения можно рассматривать, как операцию декартова произведения, при наличии условия ограничения на общий атрибут отношений, по которому реализовано соединение.

Отношение 1:

A	B	C	D
a	b	1	K
a	d	7	l
c	g	10	m
f	b	14	n

Отношение 2:

K	C	N
x	7	u
y	14	t

Соединение отношения 1 с отношением 2 по атрибуту C.

A	B	C	D	K	N
a	d	7	l	x	u
f	b	14	n	y	t

Атрибут, по которому производится соединение, в итоговом отношении указывается один раз.

Запрос: Вывести полную информацию по осуществляемым полетам и обеспечивающим эти полеты самолетам.

Перечисляются все атрибуты итогового отношения.

SQL 1 вариант:

```
SELECT Полет.*, Самолет.*
FROM Полет, Самолет
WHERE Полет.тип_самолета =
Самолет.тип_самолета
```

SQL 2 вариант:

```
SELECT Полет.*, Самолет.*
FROM Полет JOIN Самолет
ON полет.тип_самолета =
самолет.тип_самолета
```

PA:

R = Полет ⊳⊲ Самолет
тип_самолета=тип_самолета

Условие соединения.

Полет

Дата	№_рейса	Код экипажа	Свободные места	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
1.10	112	15	7	Боинг-7	5
2.10	47	18	4	Боинг-7	4
3.10	35	17	3	Ту-154	2

Самолет

Тип самолета	Число экипажа	Кол-во мест	Вес груза
Ту-154	5	80	5
Боинг-7	8	110	7

Результат:

Дата	№_рейса	Код экипажа	Свободные места	Тип самолета	Объем груза	Число экипажа	Кол-во мест	Вес груза
1.10	35	17	12	Ту-154	3	5	80	5
1.10	112	15	7	Боинг-7	5	8	110	7
2.10	47	18	4	Боинг-7	4	8	110	7
3.10	35	17	3	Ту-154	2	5	80	5

7.4.10 Деление

Делимое – отношение 1, включает в себя связку атрибутов вида:

A	B	C	D
	1	3	
	2	1	
	2	2	
	2	3	
	3	2	

Делитель – отношение 2, включает один из атрибутов представленной пары B-C.

K	C	N
	1	
	2	
	3	

Частное – результат операции деления, отношение1 / отношение2.

B
2

Результат – значение по атрибуту B отношения 1, включающее множество всех допустимых значений C отношения 2.

Запрос: Определить дату, когда осуществляют рейсы все возможные типы самолетов.
Признак операции деления в запросе – все возможные.

SQL:

```
SELECT DISTINCT дата FROM Полет A
WHERE NOT EXISTS
 (SELECT тип_самолета FROM Самолет
 WHERE NOT EXISTS
 (SELECT тип_самолета FROM Полет B
 WHERE B.тип_самолета = самолет.тип_самолета
AND A.дата=B.дата))
```

PA:
R = Полет / Самолет

При анализе операции деления необходимо помнить, что SQL-запросы выполняются с внутреннего уровня вложенности.

3 запрос	SELECT DISTINCT дата FROM полет A WHERE NOT EXISTS	DISTINCT исключает повторяющиеся значения при выводе результата
2 запрос	(SELECT тип_самолета FROM самолет WHERE NOT EXISTS	Формирование «дополнения» на основе существующего множества всех самолетов (отношение самолет) к множеству типов самолетов, осуществляющих полет по заданной (конкретной) дате. Если «дополнение» = ∅ – пустое множество, то в указанную дату осуществляют полет все возможные типы самолетов.
1 запрос	(SELECT тип_самолета FROM полет B WHERE B.тип_самолета = самолет.тип_самолета and A.дата=B.дата))	Формирование множества типов самолетов, осуществляющих полет в конкретную дату. Связывание двух копий таблицы полет по дате вылета.

Полет

Дата	№ рейса	Код экипажа	Свободные места	Тип самолета	Объем груза
1.10	35	17	12	Ту-154	3
1.10	112	15	7	Боинг-7	5
2.10	47	18	4	Боинг-7	4
3.10	35	17	3	Ту-154	2

Самолет

Тип самолета	Число экипажа	Кол-во мест	Вес груза
Ту-154	5	80	5
Боинг-7	8	110	7

Для реализации операции деления указанные отношения могут быть приведены к следующему виду:

Полет A

Дата	Тип самолета
1.10	Ту-154
1.10	Боинг-7
2.10	Боинг-7
3.10	Ту-154

Самолет

Тип самолета
Ту-154
Боинг-7

Полет B

Дата	Тип самолета
1.10	Ту-154
1.10	Боинг-7
2.10	Боинг-7
3.10	Ту-154

Результат операции деления:

Дата
1.10

Связывание на уровне вложенного внутреннего запроса двух копий A и B отношения полет по дате = ‘1.10’ (первый кортеж отношения полет A). Результатом первого запроса будет множество {Ту-154, Боинг-7}.

Второй запрос работает с отношением самолет, формируя «дополнительное» множество к полученному. Для даты = ‘1.10’ имеем ∅.

Третий запрос формирует результат. Если этот запрос выдает ∅, то в указанную дату использовались все типы самолетов.

Для даты = ‘1.10’ (вторая строка отношения полет A): 1 запрос – {Ту-154, Боинг-7}; 2 запрос – ∅; 3 запрос не включает дату ‘1.10’ в итоговый список повторно (работает DISTINCT).

Для даты = ‘2.10’: 1 запрос – {Боинг-7}; 2 запрос – {Ту-154}; 3 запрос не включает дату ‘2.10’ в итоговый список.

Для даты = ‘3.10’: 1 запрос – {Ту-154}; 2 запрос – {Боинг-7}; 3 запрос не включает дату ‘3.10’ в итоговый список.

7.5 Ход работы

9. Ознакомиться с операциями реляционной алгебры.
10. Применить к разработанной БД (лабораторная работа №3) операции селекции и соединения в одном запросе.
11. Создать запрос, использующий операции проекции и деления (в одном запросе).
12. Создать запрос, использующий операции проекции, объединения и конъюнкции (в одном запросе).
13. Создать запрос, использующий операции соединения и деления (в одном запросе).
14. Создать запрос, использующий операции вычитания и дизъюнкции (в одном запросе).
15. Сформулировать и записать запрос на SQL, не реализующийся на РА.

7.6 Содержание отчета

1. Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.
2. Отчет должен содержать описание действий студента по конкретному варианту.

3. Запросы должны быть сформулированы (на русском языке), представлены в форме РА и SQL.

4. Отчет должен содержать:
   - таблицу исходных данных;
   - тексты запросов;
   - результаты их выполнения.

7.7 Контрольные вопросы

1. Поясните действие операции проекции.
2. Приведите пример операции селекции.
3. Чем отличаются операции РА соединение и объединение.
4. Продемонстрируйте на примере, как выразить операцию соединения через декартово произведение.
5. Сформулировать и записать запрос на РА, не реализующийся на SQL.

ЛАБОРАТОРНАЯ РАБОТА № 8
Язык SQL. Генераторы. Триггеры

8.1 Цель работы

Выработать у обучающихся практические навыки по работе с реляционными базами данных, ознакомить с принципом работы генераторов и триггеров.

8.2 Основные положения

8.2.1 Генераторы

Генератор – объект базы данных, служащий для генерации последовательностей целых чисел. Во многих таблицах используются искусственные первичные ключи. Например, в таблице «Должность» два поля – «Номер должности» (первичный ключ) и «Наименование должности». В данном случае первичный ключ – искусственный, он не является характеристикой сущности «Должность», а был введен специально (искусственно) для того, чтобы не сносить текстовое поле в дочерние таблицы. В случае применения искусственного ключа по натуральному ключу «Наименование должности» должен быть создан альтернативный ключ.

Каждый раз, занося новую строку в таблицу «Должность», в качестве номера первичного ключа надо использовать следующее целое число. Это можно сделать двумя способами:

1. Сначала запросом получить максимальный номер должности в таблице, затем прибавить к нему единицу и затем занести новую строку.
2. Воспользоваться генератором, который подставит следующее значение автоматически.

Первый подход хоть и неудобен, но работает в однопользовательских базах. В многопользовательских базах нет гарантии, что два клиента одновременно не сгенерируют одинаковый номер. Этого недостатка лишен генератор – каждый клиент получит разные номера.

Создание генератора. Для создания генератора служит команда CREATE GENERATOR. После создания, генератор имеет значение 0. Синтаксис оператора:

```
CREATE GENERATOR name;
```

Здесь name – имя генератора. Имя генератора, как правило, содержит имя поля, для которого он предназначен. Например, генератор для поля TaskID может называться TaskID_GEN.

Для того чтобы установить генератор в другое значение, необходимо использовать команду SET GENERATOR:

```
SET GENERATOR name TO value;
```

Здесь name - имя генератора, value - новое значение генератора (число в диапазоне от -264 до 264-1).

После вызова оператора SET GENERATOR следующее значение, которое вернет функция GEN_ID() (см. пример ниже) будет равно value + инкремент, указанный в функции. Перед вызовом функции необходимо удостовериться, что после того как генератор будет переустановлен, он будет возвращать уникальные значения (если он используется для генерации первичного ключа).

Использование генераторов. Генератор представляет собой переменную для хранения некоторого текущего значения. После создания генератора место под переменную отведено, генератору присвоено начальное значение. Для того чтобы получить следующее значение генератора, необходимо применить функцию GEN_ID (здесь name – имя генератора, step – инкремент генератора):

```
GEN_ID( name, step);
```

Функция GEN_ID может быть использована для ввода значений в таблицу, в триггере или хранимой процедуре. Пример использования функции:

```
INSERT INTO Subject (SubjectID, SubjectName)
       VALUES (GEN_ID(SubjectID_GEN,1),'OBD');
```

Генератор возвращает 64-битовое значение. Столбец, в котором сохраняется значение генератора, должен быть соответствующего типа (DECIMAL или NUMERIC). В процедуре переменная для сохранения значения генератора должна быть типа ISC_INT64.

Удаление генератора. Не существует оператора DROP GENERATOR. Генератор можно удалить, удалив его из системной таблицы:

```
DELETE FROM RDB$GENERATORS WHERE RDB$GENERATORS_NAME = 'SubjectID_GEN';
```

8.2.2 Триггеры

Триггер – процедура, автоматически вызываемая при операциях с таблицей. Под операциями понимаются операторы INSERT, UPDATE, DELETE. Каждый триггер может быть вызван до соответствующей операции или после нее. Преимущества использования триггеров:

- автоматическое отслеживание целостности данных не только на уровне связи между таблицами, но и любым произвольным образом;
- облегчение написания приложений БД и их поддержки.

Синтаксис оператора CREATE TRIGGER. Оператор состоит из заголовка триггера и тела триггера. Заголовок содержит:

- имя триггера, уникальное по БД;
- имя таблицы, с которой ассоциируется триггер;
- указание на момент, когда триггер должен вызываться.

Тело триггера состоит из опционального списка локальных переменных и операторов.

Синтаксис оператора:

```
CREATE TRIGGER name FOR { table | view}
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} {DELETE | INSERT | UPDATE}
  [POSITION number]
  AS < trigger_body>
< trigger_body> = [<variable_declaration_list>] < block>
< variable_declaration_list> = DECLARE VARIABLE variable datatype;
[DECLARE VARIABLE variable datatype; .]
< block> =
BEGIN
< compound_statement> [<compound_statement> .]
END
< compound_statement> = {<block> | statement;}
```

Используемые обозначения в синтаксисе оператора приведены в таблице 8.1.

Заголовок триггера. Все, что идет до части **AS** оператора **CREATE TRIGGER** составляет заголовок триггера. Заголовок триггера должен определять имя триггера и имя ассоциированной с триггером таблицы или пользовательского представления. Таблица или пользовательское представление должны существовать на момент выполнения оператора **CREATE TRIGGER**. Оставшаяся часть оператора определяет, когда и как вызывается триггер:

- статус триггера, ACTIVE or INACTIVE. Если триггер активный, он вызывается при наступлении события триггера. Если триггер неактивный, он не вызывается.
- время вызова триггера: BEFORE (до) или AFTER (после) некоторого действия.
- операция, с которой связан триггер: INSERT, UPDATE, или DELETE. Может быть указана только одна операция. Если один и тот же триггер должен выполнятся на несколько операций, то необходимо создать несколько триггеров с одинаковым телом, различающихся именем и операцией.
- (опционально) номер триггера по порядку среди всех триггеров, ассоциированных с данной операцией на данной таблице - POSITION. Номер позиции может быть любым числом от 0 до 32767. Значение по умолчанию - 0. Триггеры с меньшими номерами вызываются раньше.

Тело триггера. Все, что следует за частью **AS** оператора **CREATE TRIGGER** составляет тело триггера. Тело триггера состоит из опционального списка локальных переменных, за которым идет блок операторов. Блок состоит из набора операторов на языке хранимых процедур и триггеров, заключенных в операторные скобки.

Таблица 8.1 – Используемые обозначения

name	Имя триггера. В имени обычно упоминают имя таблицы и момент запуска триггера. Например, WORKER_BI_T1 - триггер #1 таблицы WORKER, вызываемый перед добавлением (before insert).
table	Имя таблицы или пользовательского представления, с которым ассоциируется триггер.
ACTIVE|INACTIVE	Указывает «активность» триггера. Не активные триггеры игнорируются. По умолчанию триггер активен.
BEFORE|AFTER	Указывает момент запуска триггера до операции (BEFORE) или после операции (AFTER).
DLETE|INSERT|UPDATE	Указывает операцию, с которой связан триггер.
POSITION number	Указывает порядок срабатывания триггера. На одно и то же действие могут быть назначены несколько триггеров. В таком случае они должны различаться позициями. Триггеры вызываются в порядке увеличения позиции. Позиции триггеров не обязательно должны идти строго друг за другом (1, 2, 3) допустимы произвольные позиции (5, 25, 37). Триггеры с одинаковыми позициями вызываются по алфавиту, в соответствии с именами.
DECLARE VARIABLE var <datatype>	Описывает локальную переменную триггера. С помощью DECLARE VARIABLE можно описать только одну переменную, за оператором должна следовать точка с запятой. Var - имя локальной переменной, должно быть уникально для триггера. <datatype> - тип данных переменной
statement	Любой одиночный оператор на языке процедур и триггеров Interbase. Любой оператор за исключением BEGIN END должен заканчиваться точкой с запятой.
terminator	Разделитель, определенный оператором SET TERM. Разделитель служит для указания конца тела триггера. Используется только в isql.

Язык хранимых процедур и триггеров Interbase. Язык хранимых процедур и триггеров Interbase (ЯХПТ) является полным языком для написания хранимых процедур и триггеров. Язык включает в себя:

- операторы манипулирования данными SQL (INSERT, UPDATE, DELETE) а также оператор SELECT;
- операторы и выражения SQL, включая функции, определяемые пользователем (UDF);
- расширения SQL, включая, оператор присваивания, операторы управления последовательностью выполнения, локальные переменные, операторы отсылки сообщений, обработка исключительных ситуаций, операторы обработки ошибок.

Несмотря на то, что триггеры и хранимые процедуры служат для разных целей и используются по-разному, они используют одинаковый язык. Существуют следующие ограничения:

- контекстные переменные используются только в триггерах:
- входные и выходные параметры, а также возвращающие результат операторы SUSPEND и EXIT могут быть использованы только в хранимых процедурах.

Элементы языка хранимых процедур и триггеров приведены в таблице 8.2.

Синтаксические ошибки в триггерах. Сообщение об ошибке выглядит следующим образом (пример):

```
Statement failed, SQLCODE = -104
Dynamic SQL Error
-SQL error code = -104
-Token unknown - line 4, char 9
-tmp
```

Номер строки считается от начала оператора CREATE TRIGGER, а не от начала всех операторов. Символы считаются от левой границы. Неизвестный элемент (token) либо непосредственно является источником ошибки, либо источник ошибки находится справа от него. В случае сомнений необходимо исследовать всю строку на наличие ошибок.

Таблица 8.2 – Элементы языка

BEGIN...END	определяет блок операторов (операторные скобки). Скобка после END не нужна.
variable = expression	оператор присваивания
/* comment_text */	многострочный комментарий
EXCEPTION exception_name	вызывает исключительную ситуацию с именем exception_name, если она не обрабатывается оператором WHEN
EXECUTE PROCEDURE proc_name [var [, var ...]] [RETURNING_VALUES var [, var ...]]	Вызывает хранимую процедуру с именем proc_name, с указанными входными и выходными параметрами. Параметры процедуры должны быть локальными переменными.
FOR select_statement DO compound_statement	повторяет выполнение блока кода следующего за DO для каждой строки, возвращенной оператором select_statement
select_statement.	select_statement - обычный запрос на выборку, за исключением того, что он обязательно должен содержать часть INTO, и данная часть должна идти на последнем месте.
compound_statement	или одиночный оператор языка, или блок операторов заключенных в операторные скобки.
IF (condition) THEN compound_statement [ELSE compound_statement]	условный оператор condition - условие, выражение булевской трехзвенной логики (TRUE, FALSE, UNKNOWN). Обычно два выражения как операнды оператора сравнения.
NEW.column	контекстная переменная, содержащая новое значение столбца с именем column при выполнении операций INSERT или UPDATE.
OLD.column	контекстная переменная, содержащая старое значение столбца с именем column при выполнении операций INSERT или UPDATE.
POST_EVENT event_name	отсылает сообщение с именем event_name.
WHILE (condition) DO compound_statement	цикл с предусловием.
WHEN {error [, error .]|ANY} DO compound_statement	Оператор обработки исключительных ситуаций и ошибок. В случае если имеет место одна из ошибок перечисленных в операторе WHEN, вызывается compound_statement. Оператор WHEN должен быть последним оператором перед END тела триггера. Error: EXCEPTION exception_name, SQLCODE errcode or GDSCODE number. Ошибкой может быть - возбуждение исключительной ситуации с именем exception_name, равенство константы SQLCODE значению errcode, равенство константы GDSCODE значению number. ANY – обрабатывает любую ошибку, если она имела место.

Контекстные переменные NEW и OLD. Контекстная переменная OLD содержит текущие или старые значения элементов строки, которая обновляется или удаляется. OLD не используется для операций добавления (INSERT). Контекстная переменная NEW содержит новые значения столбцов строки, которые будут присвоены после операции добавления или обновления. NEW не имеет смысла для операции удаления. Контекстные переменные, как правило, используются для сравнения нового и старого значения столбца до его изменения и после.

Синтаксис контекстных переменных:

```
NEW.column
OLD.column
```

где column - имя столбца.

Контекстные переменные могут использоваться так же, как и простые локальные переменные.

Новое значение для столбца может быть получено только до операции. В триггере, который вызывается после операции INSERT, присвоение переменной NEW не имеет смысла. При проведении операции сначала вызываются триггеры BEFORE, они «видят» старое значение строки, затем производится действие, после чего вызываются триггеры AFTER, которые «видят» новое значение строки. Триггеры, описанные как BEFORE и обращающиеся к NEW не имеют смысла.

Например, рассмотрим триггер:

```
SET TERM !! ;
CREATE TRIGGER SAVE_SALARY_CHANGE FOR EMPLOYEE
 AFTER UPDATE AS
 BEGIN
    IF (old.salary <> new.salary) THEN
 INSERT INTO SALARY_HISTORY
(EMP_NO, CHANGE_DATE, UPDATER_ID, OLD_SALARY, PERCENT_CHANGE)
  VALUES (old.emp_no, 'now', USER, old.salary,
    (new.salary - old.salary) * 100 / old.salary);
  END!!
SET TERM ; !!
```

Тригер вызывается после обновления таблицы EMPLOYEE, и сравнивает старую и новую сумму продаж. Если сумма изменилась, триггер производит запись в таблицу SALARY_HISTORY (история продаж).

Модификация триггеров. Для модификации триггеров используется оператор ALTER TRIGGER. С помощью ALTER TRIGGER можно менять тело триггера, заголовок триггера, а также триггер целиком.

Для того чтобы изменить триггер, автоматически создаваемый СУБД для проверки условия CHECK, следует использовать оператор ALTER TABLE.

Синтаксис ALTER TRIGGER:

```
ALTER TRIGGER name
  [ACTIVE | INACTIVE]
  [{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
  [POSITION number]
AS <trigger_body>
```

Синтаксис ALTER TRIGGER похож на CREATE TRIGGER за исключением:

- CREATE заменено на ALTER;
- пропущена часть FOR <имя таблицы>, так как нельзя менять таблицу, с которой ассоциирован триггер;
- оператор должен содержать только те параметры, которые должны быть изменены, ниже приводятся исключения.

Модификация заголовка триггера. При модификации заголовка триггера ALTER TRIGGER требует, чтобы была изменена хотя бы одна позиция после имени триггера. Все характеристики триггера, пропущенные в ALTER TRIGGER сохраняют свои старые значения.

Следующий оператор делает триггер не активным.

```
ALTER TRIGGER SAVE_SALARY_CHANGE INACTIVE;
```

Если меняется момент вызова триггера (BEFORE, AFTER), то обязательно надо указать операцию (UPDATE, INSERT, DELETE).

Следующий оператор делает триггер активным, и меняет момент его запуска на BEFORE UPDATE:

```
ALTER TRIGGER SAVE_SALARY_CHANGE
ACTIVE
BEFORE UPDATE;
```

Модификация тела триггера. Заменить можно только тело триггера целиком. При этом новое описание полностью заменяет ранее существующее. При изменении тела триггера оператор может не содержать заголовочной информации за исключением имени триггера.

Последовательность действий для модификации тела триггера:

- извлечь описание триггера;
- заменить CREATE на ALTER, удалить всю заголовочную информацию после имени триггера до ключевого слова AS;
- модифицировать тело триггера, запустить оператор.

Удаление триггера. Триггер можно удалить командой DROP TRIGGER. Синтаксис оператора:

```
DROP TRIGGER <имя триггера>
```

Триггеры и транзакции. Триггеры работают в контексте транзакции вызвавшего их приложения. В случае если транзакция приложения будет отменена, будут отменены и все действия триггера.

Триггеры и сообщения о событиях. Триггеры могут быть использованы для того, чтобы уведомить приложение о том, что произошло некоторое событие. Для примера рассмотрим триггер POST_NEW_ORDER, который генерирует событие с именем «NEW_ORDER» в случае если добавляется запись в таблицу SALES.

```
SET TERM !! ;
CREATE TRIGGER POST_NEW_ORDER FOR SALES
  AFTER INSERT AS
  BEGIN
      POST_EVENT 'NEW_ORDER';
  END !!
SET TERM;
```

В общем случае триггер может использовать переменную в качестве имени события:

```
POST_EVENT :EVENT_NAME;
```

8.2.3 Использование триггеров для обновления пользовательских представлений

Пользовательские представления, основанные на соединении нескольких таблиц, как правило, не обновляемы, и могут служить только для чтения. Тем не менее, можно написать триггеры для операций добавления, обновления и удаления таким образом, что они будут правильно модифицировать базовые таблицы, из которых «собран» VIEW. Таким образом, можно редактировать не редактируемые пользовательские представления.

Следующие операторы создают две таблицы, создают пользовательское представление и три триггера для модификации представления:

```
CREATE TABLE Table1 (
  ColA INTEGER NOT NULL,
  ColB VARCHAR(20),
  CONSTRAINT pk_table PRIMARY KEY(ColA)
);

CREATE TABLE Table2 (
  ColA INTEGER NOT NULL,
  ColC VARCHAR(20),
  CONSTRAINT fk_table2 FOREIGN KEY (ColA)
                  REFERENCES Table1(ColA)
);

CREATE VIEW TableView AS
  SELECT Table1.ColA, Table1.ColB, Table2.ColC
      FROM Table1, Table2
    WHERE Table1.ColA = Table2.ColA;
```

Создание триггера, который позволит производить удаление данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Delete FOR TableView BEFORE DELETE AS
  BEGIN
      DELETE FROM Table1
        WHERE ColA = OLD.ColA;
      DELETE FROM Table2
        WHERE ColA = OLD.ColA;
  END;
```

Создание триггера, который позволит производить обновление данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Update FOR TableView
BEFORE UPDATE AS
  BEGIN
      UPDATE Table1
        SET ColB = NEW.ColB
        WHERE ColA = OLD.ColA;
      UPDATE Table2
        SET ColC = NEW.ColC
        WHERE ColA = OLD.ColA;
  END;
```

Создание триггера, который позволит производить выполнять ввод данных в таблицах Table1, Table2:

```
CREATE TRIGGER TableView_Insert FOR TableView BEFORE INSERT AS
  BEGIN
      INSERT INTO Table1 values (NEW.ColA,NEW.ColB);
      INSERT INTO Table2 values (NEW.ColA,NEW.ColC);
  END;
```

8.2.4 Использование исключений в триггерах

Исключение – это именованное сообщение об ошибке, которое может быть вызвано из триггера или хранимой процедуры. Исключения создаются с помощью оператора CREATE EXCEPTION, модифицируется с помощью ALTER EXCEPTION, и удаляются с помощью DROP EXCEPTION.

Исключения - это объекты базы данных, они хранятся в системной таблице и могут использоваться любой процедурой или триггером.

Если исключение вызывается в хранимой процедуре или триггере, исключение возвращает строковое сообщение вызывающей программе и завершает выполнение блока операторов, если исключение не обрабатывается с помощью оператора WHEN ниже в тексте процедуры или триггера.

Например, если количество неудачных попыток зайти в систему (каждая попытка записывается в таблицу) превышает 5, можно выводить предупреждающее сообщение.

Генерация исключения. Для генерации исключения в триггере нужно воспользоваться следующим оператором: **EXCEPTION** name; здесь name - это имя существующего исключения.

Пример создания исключения:

```
CREATE EXCEPTION RAISE_TOO_HIGH 'New salary exceeds old
            by more than 50%. Cannot update record.';
```

Пример вызова исключения из триггера:

```
SET TERM !! ;
CREATE TRIGGER SAVE_SALARY_CHANGE
FOR EMPLOYEE
  AFTER UPDATE AS
  DECLARE VARIABLE PCNT_RAISE;
BEGIN
  PCNT_RAISE=(NEW.SALARY-OLD.SALARY)*100/OLD.SALARY;
      IF (OLD.SALARY <> NEW.SALARY)
      THEN IF (PCNT_RAISE > 50)
            THEN EXCEPTION RAISE_TOO_HIGH;
      ELSE BEGIN
            INSERT INTO SALARY_HISTORY (EMP_NO, HANGE_DATE,
                UPDATER_ID, OLD_SALARY, PERCENT_CHANGE)
            VALUES (OLD.EMP_NO, 'NOW', USER, OLD.SALARY, PCNT_RAISE);
              END
   END !!
SET TERM ; !!
```

8.3 Порядок выполнения лабораторной работы

1. В соответствии с вариантом задания создать генератор и триггер (см. приложение Д).
2. Изменить значение генератора, в соответствии с хранимыми данными.
3. Ввести данные в таблицу, используя генератор (не менее 5 строк). Просмотреть полученный результат.
4. Внести изменения в указанные таблицы, используя триггеры (не менее 5 строк). Просмотреть полученный результат.
5. Написать отчет.

8.4 Содержание отчета

Отчет состоит из титульного листа, цели работы, описания процесса выполнения работы и вывода.

Отчет должен содержать исходные данные, этапы работы по созданию и изменению генератора, по созданию представлений и триггеров; ввод данных в таблицу, с использованием генератора и триггера, выборку информации из таблицы на каждом шаге работы.

8.5 Контрольные вопросы

1. Назначение генераторов.
2. Как сгенерировать следующее значение генератора?
3. Как переустановить значение генератора?
4. Как удалить генератор?
5. Повышение надежности данных.
6. Организация многопользовательского режима доступа к данным.
7. Что такое «триггер»?
8. Из каких частей состоит триггер?
9. Какая информация содержится в заголовочной части триггера?
10. Как сделать триггер временно неактивным? Как удалить триггер?
11. Для чего используются триггеры?
12. Назовите элементы языка хранимых процедур и триггеров.

ЛАБОРАТОРНАЯ РАБОТА № 9
Проектирование реляционных баз данных. Нормализация отношений

9.1 Цель работы

Осуществление исследования и анализа предметной области, построение диаграммы «сущность-связь» и модели данных, основанной на ключах.

9.2 Основные положения

Основная цель процесса проектирования БД состоит в получении такого проекта, который удовлетворяет следующим требованиям:

1) корректность схемы БД, т.е. база должна быть гомоморфным образом моделируемой предметной области (ПрО), где каждому объекту предметной области соответствуют данные в памяти ЭВМ, а каждому процессу – адекватные процедуры обработки данных;

2) обеспечение ограничений (на объёмы внешней и оперативной памяти и другие ресурсы вычислительной системы);

3) эффективность функционирования (соблюдение ограничений на время реакции системы на запрос и обновление данных);

4) защита данных (от аппаратных и программных сбоев и несанкционированного доступа);

5) простота и удобство эксплуатации;

6) гибкость, т.е. возможность развития и адаптации к изменениям предметной области и/или требований пользователей.

Удовлетворение требований 1-4 обязательно для принятия проекта.

Максимально упростить и формализовать процессы формирования требований и проектирования системы позволяют современные CASE-средства.

9.2.1 Основы проектирования баз данных

В настоящее время при разработке информационных систем используются специальные программные средства – CASE-средства, реализующие CASE-технологию создания и сопровождения информационных систем. Значение термина CASE охватывает процесс разработки и сопровождения сложных систем в целом.

CASE-технология представляет собой методологию проектирования информационных систем, набор методов, нотаций (установленные способы отображения элементов системы, т.е. графы, таблицы, блок-схемы, формальные и естественные языки) и инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать модель системы на всех этапах разработки и сопровождения системы и разрабатывать приложения в соответствии с информационными потребностями пользователей.

Процесс проектирования БД представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели. В общем случае процедуру проектирования базы данных разбивают на три этапа, каждый из которых завершается созданием соответствующей информационной модели.

1. Концептуальное проектирование – создание представления (схемы, модели) БД, включающего определение важнейших сущностей (таблиц) и связей между ними, но не зависящего от модели БД (иерархической, сетевой, реляционной и т.д.) и физической реализации (целевой СУБД).

2. Логическое проектирование – развитие концептуального представления БД с учетом принимаемой модели (иерархической, сетевой, реляционной и т.д.).

3. Физическое проектирование – развитие логической модели БД с учетом выбранной целевой СУБД**.**

Концептуальное и логическое проектирование вместе называют также *инфологическим* или *семантическим проектированием*.

В настоящее время для проектирования БД активно используются CASE-средства, в основном ориентированные на использование ERD (Entity – Relationship Diagrams, диаграммы «сущность-связь»), являющейся информационной моделью. С их помощью определяются важные для предметной области объекты (сущности), отношения друг с другом (связи) и их свойства (атрибуты). Средства проектирования ERD в основном ориентированы на реляционные базы данных (РБД), и если существует необходимость проектирования другой системы, например, объектно-ориентированной, то лучше избрать другие методы проектирования.

ERD были впервые предложены П. Ченом в 1976 г. Этапы и правила построения диаграммы «сущность-связь» в нотации П.Чена рассмотрены в п. 3.2.1 методических указаний по выполнению курсовой работы по дисциплине «Управление данными».

При использовании любого CASE-средства вначале строится логическая модель БД в виде диаграммы с указанием сущностей и связей между ними, а тем физическая модель.

Логической моделью данных называется универсальное представление структуры данных, независимое от конечной реализации базы данных и аппаратной платформы. Логическая модель позволяет понять суть проектируемой системы, отражая логические взаимосвязи между сущностями. На основании полученной логической модели переходят к физической модели данных.

Физическая модель представляет собой диаграмму, содержащую всю необходимую информацию для генерации БД для конкретной СУБД или даже конкретной версии СУБД и отражает физические свойства проектируемой БД (типы данных, размер полей, индексы). Параметры физической информационной модели зависят от выбранной СУБД. Если в логической модели не имеет значения, какие идентификаторы носят таблицы и атрибуты, тип данных атрибутов и т.д., то в физической модели должно быть полное описание БД в соответствии с принятым в ней синтаксисом, с указанием типов атрибутов, триггеров, хранимых процедур и т.д. По одной и той же логической модели можно создать несколько физических.

Такой порядок действий называется *прямое проектирование БД (Forward Engineering DB)*. CASE-средства позволяют выполнять также *обратное проектирование БД (Reverse Engineering DB)*, т.е. на основании системного каталога БД или DDL-скрипта (Data Definition Language – язык описания данных) построить физическую и, далее, логическую модель данных.

9.2.2 Типы структур баз данных

Наиболее распространены три типа структур и моделей данных: иерархическая или древовидная; сетевая; реляционная (плоские файлы).

Если сетевая структура имеет связи типа M:M, то ее называют *сложной сетевой структурой (ССС)* в противном случае – *простой сетевой структурой (ПСС)*.

Преобразование сложной сетевой структуры в простую сетевую структуру. Сложную сетевую структуру можно преобразовать в простую сетевую, введя дополнительный тип записи. Этот тип записи должен содержать ключевые атрибуты объектов, участвующих в связях (см. Рисунок 9.1).

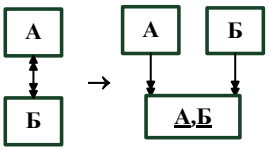


Рисунок 9.1 – Схема преобразования ССС в ПСС

Преобразование простой сетевой структуры в древовидную структуру. Любую простую сетевую структуру можно преобразовать в древовидную, введя избыточность посредством дублирования отношения, входящего в ПСС-связь (1:M) со стороны M. На Рисунок 9.2 приведены примеры таких преобразований.

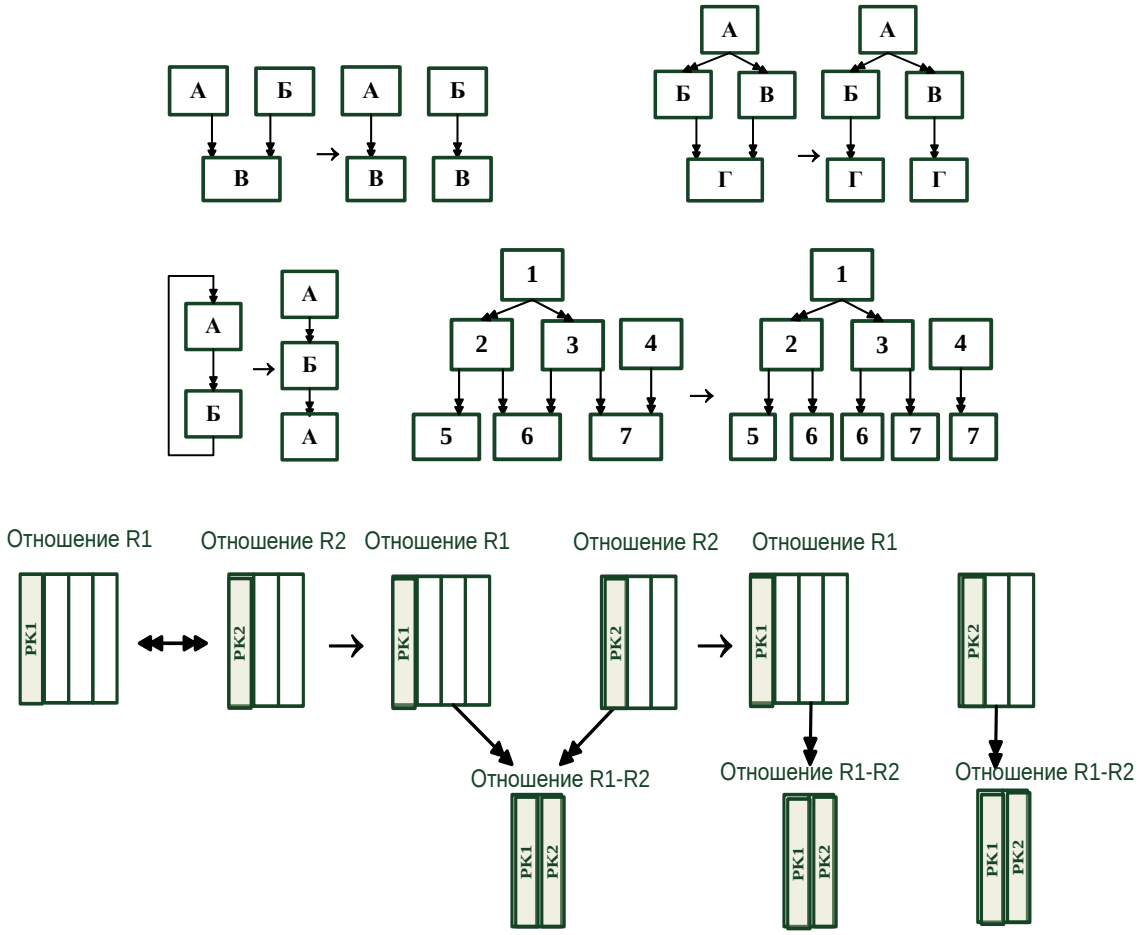


Рисунок 9.2 – Примеры преобразования ПСС в древовидные структуры

После составления концептуальной (логической) схемы БД необходимо проверить её на отсутствие аномалий модификации данных. При неправильно спроектированной схеме БД могут возникнуть аномалии выполнения операций модификации данных. Эти аномалии обусловлены ограниченностью структуры реляционной модели данных.

Аномалия – такая ситуация в таблице БД, которая приводит к противоречию в БД, либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов.

Различают три вида аномалий: аномалии обновления, удаления и добавления. Аномалия обновления может возникнуть в том случае, когда информация дублируется. Другие аномалии возникают тогда, когда две и более сущности объединены в одно отношение.

Виды аномалий модификации данных

− аномалии вставки (INSERT) – возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы (например, нельзя добавить информацию о поставщике, который не поставил ни одного товара; или нельзя добавить информацию о преподавателе, если данный предмет не выбрал ни один студент и т.п.);

– аномалии обновления (UPDATE) – проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы (например, эта проблема возникнет в том случае, если необходимо переместить поставщика из одного города в другой; или чтобы изменить товар «сода» на «сода пищевая», нужно просмотреть всю таблицу);

– аномалии удаления (DELETE) – при удалении какого-либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью (например, с удалением поставки исчезнет информация о поставщике).

Для решения проблемы аномалии модификации данных при проектировании реляционной БД проводится нормализация отношений.

Табличные структуры данных (реляционная модель). Понятие РМД (реляционной модели БД) основывается на:

- целостности отношений – связей по внешним ключам;
- нормализации отношений – процессе оптимизации схемы с целью повышения эффективности манипулирования данными.

Процесс приведения древовидной структуры (ДС) к табличной форме называется *нормализацией.* Это понятие ввел Кодд.

Задача преобразования древовидной структуры к табличной форме состоит в ведении избыточности. Существуют 7НФ. Приведение к каждой из последующих нормальных форм связано с устранением избыточности посредством анализа зависимостей отношения.

*Примечание**: введение избыточности в процессе формирования РБД из ДС, а затем ее устранение в процессе нормализации не может быть описано операциями сложения и вычитания. В данном случае используются реляционные операции соединения и проекции, а это иные операции.*

9.2.3 Нормализация отношений

Под *нормализацией* отношения подразумевается процесс приведения отношения к одной из так называемых *нормальных форм* (или в дальнейшем НФ). Но сначала определим зачем же нужна нормализация.

База данных – это не просто хранилище фактов (с этой задачей способны справиться и незатейливые плоские файлы). При проектировании баз данных упор в первую очередь делается на достоверность и непротиворечивость хранимых данных, причем эти свойства не должны утрачиваться в процессе работы с данными, т.е. после многочисленных изменений, удалений и дополнений данных по отношению к первоначальному состоянию БД.

Для поддержания БД в устойчивом состоянии используется ряд механизмов, которые получили обобщенное название *средств поддержки целостности*. Эти механизмы применяются как статически (на этапе проектирования БД), так и динамически (в процессе работы с БД). Рассмотрим ограничения, которым должна удовлетворять БД в процессе создания, независимо от ее наполнения данными. Приведение структуры БД в соответствие этим ограничениям – это и есть нормализация.

В целом суть этих ограничений весьма проста: каждый факт, хранимый в БД, должен храниться один-единственный раз, поскольку дублирование может привести (и на практике непременно приводит, как только проект приобретает реальную сложность) к несогласованности между копиями одной и той же информации. Следует избегать любых неоднозначностей, а также избыточности хранимой информации.

Впрочем, для устранения всяческих неопределенностей и неоднозначностей традиционно используется формальный язык математики. Итак, наша цель – приведение отношений к НФ. Следует заметить, что в процессе нормализации постоянно встречается ситуация, когда отношение приходится разложить на несколько других отношений. Поэтому более корректно было бы говорить о нормализации не отдельных отношений, а всей их совокупности в БД. В примерах для простоты рассмотрены отдельные отношения, нормализация же реальных баз данных – гораздо более трудоемкий процесс.

Цель нормализации – получение такого проекта базы данных, в котором каждый факт появляется лишь в одном месте (исключена избыточность информации).

Нормализация может не представлять такую уж проблему, если БД проектируется сразу по определенным канонам. Другими словами, можно сначала сделать БД как попало, а потом нормализовать ее, или же с самого начала строить ее по правилам, чтобы в дальнейшем не пришлось переделывать. Сейчас мы пойдем первым путем, т.е. возьмем в качестве примеров никуда не годные БД и попытаемся привести их в порядок. При этом будем иметь в виду, что при профессиональном подходе к проектированию БД используется одна из хорошо проработанных технологий (например, IDEF1X), и, возможно, какие-либо инструментальные средства, поддерживающие эту технологию.

Что же представляет собой процесс нормализации? Фактически это не что иное, как последовательное преобразование исходной БД к НФ, при этом каждая следующая НФ обязательно включает в себя предыдущую, т.е. при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются (что, собственно, и позволяет разбить процесс на этапы и производить его однократно, не возвращаясь к предыдущим этапам). Всего в реляционной теории насчитывается 7 НФ: 1-я НФ (обычно обозначается также 1НФ), 2НФ, 3НФ, 4НФ, Бойса-Кодда (НФБК), 4НФ, 5НФ.

На практике, как правило, ограничиваются 3НФ, ее оказывается вполне достаточно для создания надежной схемы БД. НФ более высокого порядка представляют скорее академический интерес из-за чрезмерной сложности. Более того, при реализации абстрактной схемы БД в виде реальной базы иногда разработчики вынуждены сделать шаг назад – провести денормализацию с целью повышения эффективности, ибо идеальная с точки зрения теории структура может оказаться слишком накладной на практике.

Первая нормальная форма (1НФ). Схема отношения **R** находится в 1НФ, если значения **dom(A)** являются атомарными для каждого атрибута **A** в **R**.

Другими словами, каждый атрибут отношения должен хранить одно-единственное значение и не являться ни списком, ни множеством значений.

Тем не менее, несмотря на внешнюю строгость данного определения, однозначно определить понятие атомарности зачастую оказывается довольно затруднительно, если заранее неизвестны семантика атрибута и его роль в обработке хранимых данных. Атрибут, который является атомарным в одном приложении, может оказаться составным в другом.

Простейший пример: в БД отдела кадров предприятия в таблице, хранящей личные сведения о сотрудниках, имеется атрибут «домашний-адрес», в котором адрес хранится в формате: город, улица, дом[/корпус], [квартира] (следуя общепринятой нотации, здесь в квадратных скобках указаны опциональные фрагменты адреса, которые могут отсутствовать). В данном случае адрес хранится в виде единой текстовой строки, поскольку маловероятно, чтобы потребовалось выбрать сотрудников, скажем, по номеру квартиры. Таким образом, в контексте БД отдела кадров адрес является атомарным понятием, и его деление на составные части не имеет смысла, т.к. только внесет в БД излишнюю громоздкость.

Однако тот же адрес для приложения, предназначенного для сортировки почты в почтовом отделении, атомарным не является, поскольку желательно сгруппировать конверты в отдельные стопки по улицам, так как каждую улицу обслуживает свой почтальон. Кроме того, с целью оптимизации перемещений почтальона в пределах улицы, каждую стопку желательно отсортировать по номерам домов, чтобы сделать возможным разнести почту за один проход по улице без возврата.

Итак, очевидно, что в отрыве от контекста затруднительно дать строгое определение атомарности, за исключением самых простых случаев (например, домен натуральных чисел). В большинстве случаев необходимо располагать сведениями о предметной области, а также о том, каким образом планируется обрабатывать хранимые в БД данные.

Приведение отношения к 1НФ – довольно простая операция. Необходимо просмотреть схему отношения и разделить составные атрибуты на различные строки/столбцы. Возможно, эту операцию придется повторить несколько раз до тех пор, пока каждый из атрибутов не станет атомарным.

Чтобы отношение соответствовало 1НФ необходимо привести это отношение в соответствие трем условиям:

1) определить первичные ключи;

2) устранить повторяющиеся группы (одинаковые кортежи);

3) привести поля отношения к атомарности.

Основой этого этапа нормализации является определение ключей.

На **Ошибка! Источник ссылки не найден.**8 раздела 2 приведена схема определения ключевых атрибутов в древовидной форме, которая называется *правилом распространения ключей.*

Вторая нормальная форма (2НФ). Перед тем, как дать определение 2НФ, необходимо познакомиться с понятием *полной* и *частичной* функциональной зависимостей (ФЗ).

Пусть **F** – множество функциональных зависимостей, при этом ФЗ **X→Y F+**. Множество **Y** называется *частично зависимым* от **X** относительно **F**, если ФЗ **X→Y** не является редуцированной слева (т.е. существует собственное подмножество **X** множества **X**, такое, что ФЗ **X'→Y F+**. Если же ФЗ **X→Y** является редуцированной слева, то множество **Y** называется *полностью зависимым* от **X** относительно **F.** Множество атрибутов **X** называется *детерминантом функциональной зависимости*, а множество атрибутов **Y** называется *зависимой частью*.

Атрибут B отношения ***функционально*** *зависит от атрибута A того же отношения (атрибуты могут быть составными) в том и только в том случае, когда в любой заданный момент времени для каждого из различных значений атрибута A обязательно существует только одно из различных значений атрибута B.*

Атрибут B находится в ***полной функциональной зависимости*** *от составного атрибута A, если он функционально зависит от A и не зависит функционально от любого подмножества атрибута A.*

Для изображения ФЗ используется графическое изображение ФЗ, так называемая *диаграмма ФЗ (или схема ФЗ)*, например, см. Рисунок 9.3.

Рисунок 9.3 – Диаграмма функциональной зависимости (или схема ФЗ)

Обозначение функциональной зависимости: $f : A \to B \Rightarrow A$ определяет B.

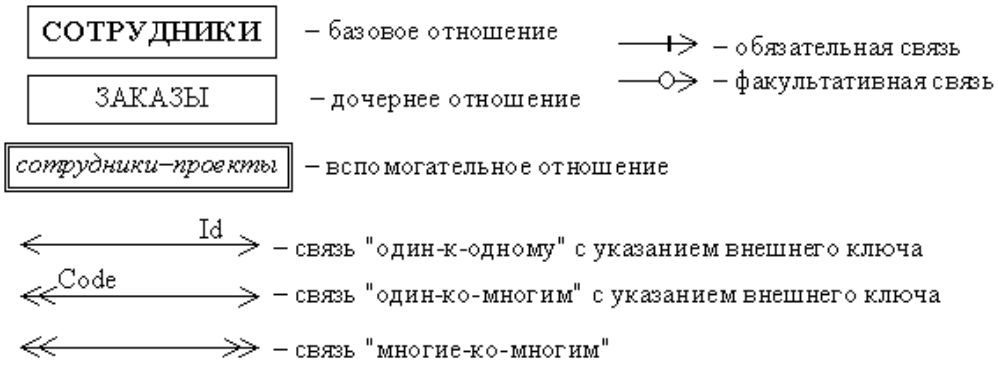


Схема отношения **R** находится во 2НФ относительно множества функциональных зависимостей **F**, если она находится в 1НФ и каждый неключевой атрибут полностью зависит от каждого ключа для **R.**

Другими словами, *отношение находится во 2НФ, если оно находится в 1НФ, и каждый неключевой атрибут характеризуется полной функциональной зависимостью от первичного ключа (т.е. все неключевые атрибуты зависят только от ключа целиком, а не от какой-то его части).*

Например, таблица «Оценки по экзаменам» характеризуется следующим набором атрибутов: Номер зачетной книжки, Дисциплина, Дата сдачи, ФИО студента, № группы, Оценка. Очевидно, что первичным ключом является набор: Номер зачетной книжки, Дисциплина, Дата сдачи. Полной функциональной зависимостью обладает только один неключевой атрибут «Оценка». Атрибуты «ФИО студента» и «№ группы» могут быть

однозначно определены по части первичного ключа – «Номер зачетной книжки». Таким образом, требуется разбиение исходной таблицы на две (см. Рисунок 9.).

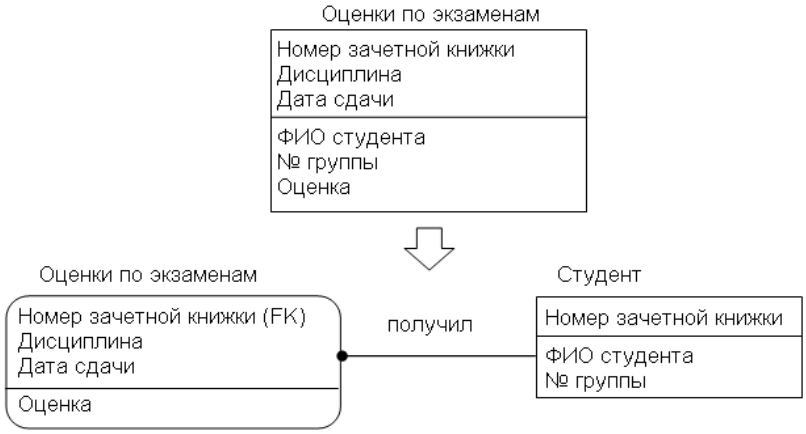


Рисунок 9.4 – Обеспечение полной функциональной зависимости

Третья нормальная форма (3НФ). Для того, чтобы формально определить 3НФ, необходимо предварительно познакомиться с понятием *транзитивной зависимости* атрибутов, от которой нужно будет попытаться избавиться на этом этапе.

Обозначим: **R** – схема отношения, **X** – подмножество **R**, **A** – атрибут в **R**, **F** – множество функциональных зависимостей. **A** называется *транзитивно зависимым* от **X** в **R**, если существует такое **Y**, являющееся подмножеством **R**, что:

- $X \rightarrow Y$
- $\sim(Y \rightarrow X)$
- $Y \rightarrow A$
- $\sim(A \ XY)$

Атрибут B отношения ***транзитивно функционально*** *зависит от атрибута A того же отношения (атрибуты могут быть составными) в том и только в том случае, если существует такой атрибут C, что имеются функциональные зависимости между атрибутами A и C, а также между B и C.*

Схема отношения **R** находится в 3НФ относительно множества функциональных зависимостей **F**, если она находится во 2НФ и ни один из непервичных атрибутов в **R** не является транзитивно зависимым от ключа для **R**.

Другим словами, *отношение находится в 3НФ, если оно находится во 2НФ и никакой неключевой атрибут функционально не зависит от другого неключевого атрибута, т.е. нет транзитивных зависимостей.*

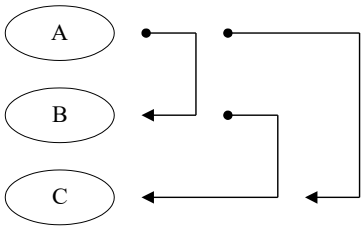
В реляционной теории имеется лемма, которая гласит, что любая схема отношения, находящаяся в 3НФ относительно **F**, находится в 2НФ относительно **F**.

Теорема Хита дает возможность выполнить декомпозицию отношений без потерь информации:

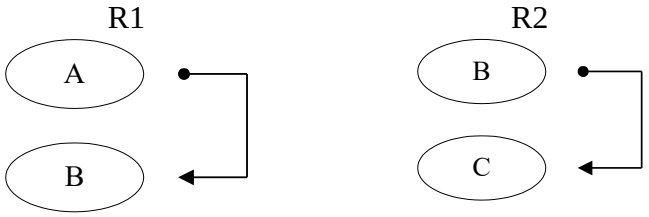
– исходное отношение с ФЗ преобразуется в другие отношения, в каждом из которых атрибуты минимально зависят от первичного ключа;

– атрибут C минимально зависит от атрибута A, если выполняется минимальная слева ФЗ $B \rightarrow C$.

Пусть A, B, C – три атрибута или три набора атрибутов отношения R. Зависимости между ними изобразим на диаграмме.



Если A → B и B → C, но B !→ A (B не является ключом), то A → C. В этом случае C транзитивно зависит от A. Преобразование в 3НФ состоит в декомпозиции исходного отношения на два отношения.



Например, таблица «Работник» характеризуется набором атрибутов: Табельный номер, Фамилия, Имя, Отчество, Должность, Зарплата, …, первичный ключ – Табельный номер. В этой таблице от первичного ключа («Табельный номер») зависит неключевой атрибут «Должность», а от «Должности» другой неключевой атрибут «Зарплата». Для приведения к 3NF необходимо добавить новую таблицу (см. Рисунок 9.).

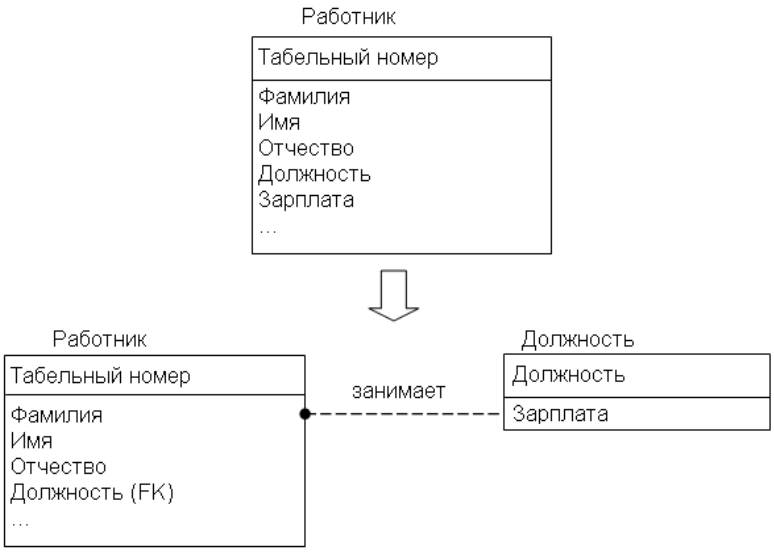


Рисунок 9.5 – Устранение транзитивной зависимости

Таким образом, чтобы привести отношение к 3НФ, необходимо устранить функциональные зависимости между неключевыми атрибутами отношения. То есть, факты, хранимые в таблице, должны зависеть только от ключа.

Нормальная форма Бойса-Кодда (НФБК). Нормальная форма Бойса-Кодда считается уточнением 3НФ. Она учитывает все потенциальные ключи, которые входят в отношения. Если отношение имеет единственный потенциальный ключ, то 3НФ и НФБК – эквивалентны. Считается, что отношение, находящееся в НФБК, если каждый его детерминант является потенциальным ключом. Чтобы убедиться, что отношение находится в НФБК необходимо отыскать все его детерминанты и убедиться, что они являются потенциальными ключами.

Можно считать БКНФ как особый случай 3НФ. На самом деле большинство таблиц соответствуют требованиям БКНФ, если они соответствуют требованиям 3НФ. Однако рассмотрим случай, если неключевой атрибут является детерминантом ключевого атрибута.

Такое обстоятельство не нарушает условий ЗНФ, но не отвечает правилам БКНФ, поскольку БКНФ требует, чтобы каждый детерминант в таблице был потенциальным ключом.

Описанную выше ситуацию, (таблица, приведенная к ЗНФ, не соответствует требованиям БКНФ), проще объяснить с помощью Рисунок 9., на котором прослеживаются следующие функциональные зависимости:

$$A + B \rightarrow C, D$$
$$C \rightarrow B$$

В структуре сущности, представленной на Рисунок 9., нет ни частичных зависимостей, ни транзитивных зависимостей (условие C → B указывает на то, что неключевой атрибут определяет часть первичного ключа – а эта зависимость не транзитивна!). Очевидно, что структура сущности, представленная на Рисунок 9. отвечает требованиям ЗНФ. А вот условие C → B нарушает требования БКНФ.

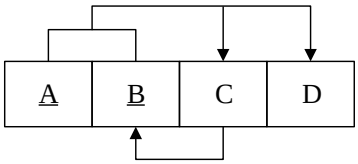


Рисунок 9.6 – Структура сущности, приведенной к ЗНФ, но не к БКНФ

Чтобы преобразовать структуру, представленную на Рисунок 9., в сущность, отвечающую требованиям как ЗНФ, так и БКНФ, прежде всего, необходимо изменить первичный ключ на A + C. Это целесообразно сделать, поскольку зависимость C → B означает, что C, по сути дела, является подмножеством B. После этого можно считать, что сущность приведена к 2НФ, поскольку в ней имеется частичная зависимость C → B. Затем, с помощью знакомых процедур декомпозиции, приведем сущность к виду, представленному на Рисунок 9.7.

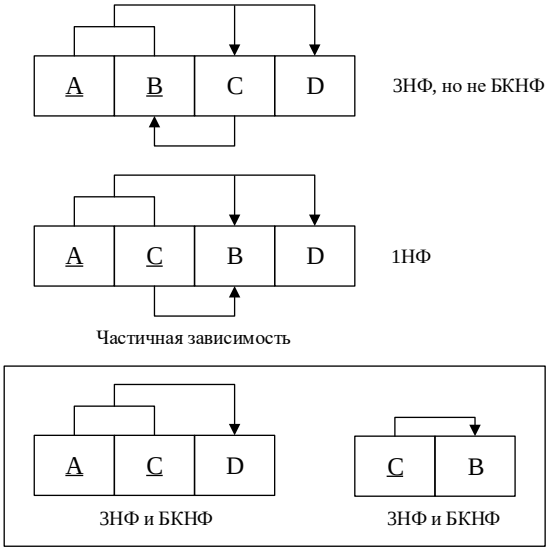


Рисунок 9.7 – Декомпозиция структуры таблицы для выполнения требований БКНФ

Примечание: для отношений, имеющих один потенциальный ключ (первичный), НФБК является 3НФ.

Четвертая нормальная форма (4НФ). В ходе проектирования БД выявлен один тип зависимости – ***многозначная зависимость***. Многозначные зависимости выявляет проблемы и избыточностью данных.

Отделение – Сотрудник – Клиент

N_отдела	ФИО_сотрудника	ФИО_клиента
011	Кот	Чижик
012	Крот	Лебедев
011	Кот	Гусев
012	Крот	Тупик

В данном отношении имеются многозначные зависимости типа «один ко многим» (1:N).

N_отдела – ФИО_клиента (1:N)

N_отдела – ФИО_сотрудника (1:N)

Если для каждого атрибута A имеется набор атрибутов B и C. Хотя атрибут B и C не зависят друг от друга.

Многозначная зависимость A → B и A → C.

Многозначная зависимость подразделяется на *тривиальную* и *нетривиальную* зависимости. Многозначная зависимость A и B, определенных на некотором отношении R, называется ***тривиальной***, если атрибут B является подмножеством атрибут A. В противном случае тривиальная зависимость является не тривиальной.

Отношение находится в 4НФ, если оно удовлетворяет НФБК и не содержит многозначных нетривиальных зависимостей.

R1: Отделение – Сотрудник (N_отделения, ФИО_сотрудника)

R2: Отделение – Клиент (N_отделения, ФИО_клиента)

Бывают случаи, когда необходимо выполнять декомпозицию на более чем два отношения. В этом случае необходимо учитывать зависимость соединения. Зависимость соединения – это свойство декомпозиции, которая вызывает генерацию ложных строк при обратном соединении декомпозированных отношений. Чтобы не возникало зависимостей соединения, необходимо отношение приводить к 5НФ.

Примечание: приведение базы данных к 4НФ сокращает дублирование данных, но появление новых отношений усложняет схему базы данных.

Пятая нормальная форма (5НФ). Отношение в 5НФ – это отношение без зависимостей соединения.

Например:

Объект – Мебель – Поставщик

N_объекта	Мебель	N_поставщика
31	Стол	P1
31	Стул	P2
52	Стул	P3
52	Кровать	P1

Для того, чтобы отношение удовлетворяло 5-ой НФ, необходимо его разбить на следующие отношения:

Объект – Мебель (N_объекта, Мебель)

Поставщик – Мебель (N_поставщика, Мебель)

Объекта - Поставщик (N_объекта, N_поставщика)

Этапы нормализации базы данных:

Шаг 1 (приведение к 1НФ). Задается одно или несколько отношений, отображающих понятия предметной области. По модели предметной области (не по внешнему виду полученных отношений!) выписываются обнаруженные функциональные зависимости. Все отношения автоматически находятся в 1НФ.

Шаг 2 (приведение к 2НФ). Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводим декомпозицию этих отношений на несколько отношений согласно процедуре приведения ко 2НФ.

Шаг 3 (приведение к 3НФ). Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов других неключевых атрибутов, то проводим декомпозицию этих отношений согласно процедуре приведения к 3НФ.

9.2.4 Денормализация отношений

Создание нормализованных отношений является важнейшей частью проектирования БД. В хорошем проекте БД должны учитываться различные требования к обработке информации. По мере того как сущности преобразуются в соответствии с требованиями нормализации, их число растет. Большее число сущностей влечет за собой увеличение количества операций обмена данными с диском (операции ввода/вывода) и увеличение времени на обработку логических связей, что приводит к снижению скорости всей системы в целом. Поэтому противоречие между эффективностью проекта, информационными потребностями и скоростью обработки информации чаще всего разрешается поиском компромисса, который, как правило, состоит в некоторой денормализации. Денормализация приводит к понижению уровня формы (т.е. в процессе денормализации 3НФ конвертируется в 2НФ). Однако необходимо помнить, что повышение производительности посредством денормализации приведет к повышению уровня избыточности данных.

Нормальные формы низких уровней имеют место в специализированных БД, называемых информационными хранилищами (data warehouses). В таких специализированных БД находят отражение все возрастающие потребности широты охвата и глубины поиска информации, на которых основаны системы поддержки решений. В информационных хранилищах, как правило, используются структуры 2НФ в условиях сложной, многоуровневой информационной среды, питающейся от множества источников.

Это не означает, что сущности рабочих баз данных, в которых содержатся частичные или транзитивные зависимости, можно просто оставить в таком виде. Ненормализованные сущности в рабочей БД, кроме возможных аномалий данных, имеют и другие недостатки:

– невысокая эффективность обновления информации, поскольку программы, которые считывают и обновляют данные, работают с сущностями больших размеров;

– затруднен процесс индексирования. Просто непрактично формировать все индексы для множества атрибутов, расположенных в единственной ненормализованной сущности;

– ненормализованные сущности затрудняют стратегии создания представлений.

Таким образом, увеличивая путём денормализации скорость обработки данных, снижается эффективность обновления сущностей (они становятся больше), индексирование усложняется и появляется угроза избыточности данных, которая может стать причиной аномалий данных. Вывод – денормализацию следует применять с осторожностью.
Обоснованием денормализации может служить требование обеспечения определённой производительности для критических запросов.

Денормализация бывает нескольких видов:

– восходящая – подразумевает перенос некоторой информации из подчинённого отношения в родительское;

– нисходящая – в этом случае информация переносится из родительского отношения в подчинённое.

Обычно денормализация применяется в случае, когда запись имеет большую длину за счёт наличия атрибутов большого объёма (графические данные, текстовые описания и проч.). Если данные этих атрибутов редко используются, то можно выделить в отдельное отношение атрибуты большого объёма. Для связи с исходным отношением вводится уникальный внешний ключ. А для получения исходного отношения создаётся представление (view), которое является соединением двух полученных отношений.

Пример нормализации отношений представлен в разделе 3.

9.3 Задание для самостоятельного выполнения

1. Произвести краткое описание предметной области (предметная область лабораторной работы соответствует варианту предметной области из курсового проекта). Подробное описание предметной области включить в раздел «Анализ предметной области» курсового проекта. Для выполнения этого этапа необходимо:
   - проанализировать информационные потребности пользователей;
   - сформировать состав документов, подлежащих включению в БД;
   - разработать состав и форму представления информации по каждому документу;
   - создать кодификаторы для упорядочения данных в БД;
   - определить задачи и функции системы.
2. Разработать первые два уровня логической модели базы данных:
   - диаграмму сущность-связь (ERD) в нотации П.Чена;
   - модель данных, основанную на ключах (KB) по методологии IDEF1.
3. Нормализовать отношения в базе данных до третьей и четвертой нормальной формы.

9.4 Контрольные вопросы

1. Этапы проектирования баз данных.
2. Инфологическое проектирование.
3. Задачи, решаемые на этапе инфологического проектирования.
4. Задачи, решаемые на этапе логического проектирования.
5. Задачи, решаемые на этапе физического проектирования
6. Сущности. Отличие понятия типа сущности и элемента сущности. Способы представления сущности.
7. Для чего предназначена диаграмма «сущность-связь»?
8. Как представляется диаграмма «сущность-связь» в нотации П. Чена?
9. Какие существуют типы связей между сущностей и чем они отличаются?
10. Какова цель модель, основанная на ключах. В чем ее особенность.
11. Атрибуты и их типы. Правила атрибутов. Классификация атрибутов.
12. Связи. Понятие безусловной, условной, биусловной, рекурсивной связи. Формализация связи. Фундаментальные виды связей. Формализация связей 1:1, 1:M, M:N.
13. Нормальные формы. Общие свойства нормальных форм. Виды нормальных форм.
14. Функциональная зависимость. Виды функциональных зависимостей. Функционально полная зависимость.
15. Функционально полная и частичная зависимости неключевого атрибута от составного ключа.
16. Определение транзитивной зависимости.
17. Условия нахождения отношений в первой нормальной форме.
18. Негативные последствия нахождение отношения лишь в первой нормальной форме.
19. Условия нахождения отношений во второй нормальной форме.
20. Условия нахождения отношений в третьей нормальной форме.
21. Условия нахождения отношений в усиленной третьей нормальной форме.
22. Многозначные зависимости.
23. Условия нахождения отношений в четвертой нормальной форме.
24. Условия нахождения отношений в пятой нормальной форме проекции-соединения.

ЛАБОРАТОРНАЯ РАБОТА № 10
Организации архитектуры «клиент-сервер» в системах баз данных

10.1 Цель работы

Построение полной атрибутивной модель базы данных в нотации IDEF1X. Изучение принципов настройка SQL-сервера на виртуальной машине под управлением операционной системы Ubuntu и по созданию сетевого взаимодействия между реальной машиной, выступающей в роли SQL-клиента и виртуальной, играющей роль SQL-сервера. Изучение механизмов организации взаимодействия с базами данных и получение практических навыков создания приложений для работы с реляционными базами данных.

10.2 Архитектура клиент-сервер для баз данных

В настоящее время большинство СУБД поддерживают режим работы клиент-сервер. Технология клиент-сервер обеспечивает прикладным программам-клиентам доступ к данным, которыми управляет сервер, и позволяет нескольким клиентам работать с одним сервером.

Сервер будем рассматривать в качестве процесса, который выполняет запросы от других процессов. Сервер базы данных будем рассматривать в качестве логического процесса, отвечающего за обработку запросов к базе данных. Клиента будем рассматривать в качестве процесса, отправляющего серверу запрос на обслуживание. К функциям клиента относятся: установление связи с сервером базы данных; запрос конкретного вида обслуживания; получение результатов запроса; подтверждение окончания обслуживания.

При использовании технологии клиент-сервер клиент посылает запрос серверу, который в соответствии с запросом выбирает данные из базы данных, возможно, подвергает их предварительной обработке и отправляет результаты клиенту. Таким образом, основную работу с базой данных выполняет сервер, что позволяет уменьшить сетевой трафик.

В качестве языка, на котором формулируются запросы к базе данных, выступает язык SQL. Основной принцип технологии клиент-сервер – разделение функций стандартного интерактивного приложения на четыре группы:

1) функции ввода и отображения данных (интерфейс);
2) прикладные функции;
3) функции хранения данных и управления информационными ресурсами;
4) служебные функции.

Прикладные функции зависят от предметной области, например, для системы продажи авиабилетов такими функциями являются поиск мест на рейс, продажа и бронирование билетов и т.д.

Служебные функции играют роль связок между функциями групп 1-3. В соответствие с этими группами выделяют три логических компонента:

1) компонент представления (для ввода и отображения данных);
2) прикладной компонент (для реализации прикладных функций);
3) компонент доступа к информационным ресурсам (для управления данными).

10.2.1 Технологии доступа к базе данных

Доступ к базе данных осуществляется с помощью интерфейса, который реализован как приложение к базе данных. Приложение может быть написано на различных языках программирования высокого уровня и с использованием различных программных средств. Для того чтобы упростить процесс разработки приложений баз данных, были созданы различные технологии, скрывающие от разработчика специфику работы с конкретной базой данных. К таким технологиям можно отнести:

– ADO (Active Data Objects) – модель программирования, которая предназначена для создания на web-серверах динамических интерактивных web-страниц для организации подключения к базам данных. Является наиболее современной технологией разработки приложений для работы с распределенными БД по технологии клиент-сервер;

– BDE (Borland Database Engine) – интерфейс между приложением и базой данных, реализованный в рамках продуктов фирмы Borland;

– ODBC (Open Database Connectivity) – технология открытого доступа к данным, предусматривает использование единого интерфейса для доступа к базам данных, поддерживающим язык SQL;

– OLE DB (Object Linking and Embedding Data Base) – технология, предоставляющее решение обеспечения COM-приложениям доступ данным независимо от типа источника данных;

– JDBC (Java Data Base Connectivity) – мобильный интерфейс к базам данных на платформе Java. Это интерфейс прикладного программирования для выполнения SQL-запросов к базам данных из программ, написанных на платформенно-независимом языке Java, позволяющем создавать как самостоятельные приложения, так и аплеты, встраиваемые в web-страницы.

Модель ADO базируется на технологии OLE DB (Object Linking and Embedding Databases – связывание и внедрение объектов баз данных). OLE является объектно-ориентированной технологией разработки повторно используемых программных компонентов. Она позволяет приложениям совместно использовать объекты, которые обладают специальными функциями. В качестве источника данных OLE-приложений могут выступать таблицы баз данных, представления, а также текстовые файлы, электронные таблицы, диаграммы и другие графические изображения.

Механизм BDE действует как интерфейс между приложением и базой данных, обеспечивая работу компонентов доступа. Он реализован как набор системных файлов DLL (Dynamic Link Library). Через BDE из приложений можно непосредственно обращаться к базам данных фирмы Borland (Paradox, dBASE), а обращение к другим базам данных BDE перенаправляет менеджеру драйверов ODBC или SQL-серверам. Технология BDE применяется в таких широко распростран енных продуктах, как C++ Builder, Borland C++, Delphi, IntraBuilder и JBuilder. Для доступа к базе данных через BDE достаточно знать алиас базы данных (т.е. имя, по которому к ней обращаются).

В стандарте ODBC язык SQL рассматривается как базовое средство доступа к данным. Этот интерфейс встраивается непосредственно в язык Си и обеспечивает высокий уровень универсальности. В результате одно и то же приложение может получить доступ к базам данных разных СУБД без внесения изменений в текст программы. Для связи приложения с любой выбранной пользователем СУБД достаточно иметь соответствующий ODBC-драйвер.

В интерфейс ODBC включены следующие элементы:

– библиотека функций, вызов которых позволяет приложению подключаться к базе данных, выполнять SQL-операторы и извлекать информацию из результирующих наборов данных;

– стандартный метод подключения и регистрации СУБД;

– стандартное представление для различных типов данных;

– стандартный набор кодов ошибок;

– типовой синтаксис SQL-операторов, построенный на использовании спецификаций стандартов X/Open и ISO CLI.

Архитектура ODBC включает четыре элемента:

1) приложение: выполняет вызов функций библиотеки ODBC для оправки SQL-операторов в СУБД и обработку возвращаемых СУБД данных;

2) менеджер драйверов: выполняет загрузку драйверов по требованию приложений. Менеджер драйверов представляет собой библиотеку DLL;

3) драйверы баз данных: обрабатывают вызовы функций ODBC и направляют SQL-запросы в конкретные источники данных, а также возвращают полученные от источников данные приложению. При необходимости драйверы выполняют модификацию запросов с целью приведения их в соответствие с синтаксическими требованиями конкретной СУБД. Драйверы предоставляют только те возможности, которые реализованы в целевой СУБД.

4) источники данных: являются базами данных, электронными таблицами и т.п. Данные в базе данных контролируются СУБД и операционной системой.

10.2.2 Объекты связи

Объект связи – объект языка программирования, осуществляющий связь между файлом данных и интерфейсом информационной системы.

Объекты связи – это объекты проекта, осуществляющие обмен информацией между интерфейсом БД и файлом данных.

Объекты связи всегда находятся на клиентской машине. Они осуществляют доступ к файлам данных, передавая информацию в интерфейс БД, и содержат внутри себя запросы, выполнения на стороне клиента.

Объекты связи также могут ограничивать доступ к информации и осуществлять защиту информации, хотя для защиты информации и ограничения доступа лучше использовать сам сервер.

Существует три технологии используемых в объектах связи:

- технология ADO;
- технология RDC;
- технология ADO.Net.

ADO является более старой технологией. Ее суть заключается в следующем: подключение к конкретной таблице или запросу, осуществляется через отдельный объект связи, т.е. все настройки и средства для работы с данными хранятся внутри конкретного объекта связи и были заложены туда при его проектировании.

Согласно технологии RDC, файлы данных рассматриваются в качестве устройств, т.е. для работ с БД нам необходим драйвер. Объект связи, работающий по технологии RDC, при работе с файлом данных сначала обращается к драйверу БД, который в свою очередь обращается к файлу данных.

Технология ADO.Net является смесью технологий ADO и RDC. Объекты связи работающие по этой технологии работают аналогично объектам, работающим по технологии ADO, однако, объекты связи входят в состав пакета Microsoft Net Framework, и автоматически обновляются вместе с этим пакетом.

Таблица 10.1 – Плюсы и минусы технологий, используемых в объектах связи

ADO	RDC	ADO.Net
+ независимость от драйверов БД, установленных в операционной системе	+ возможность работать с современными БД	+ возможность работать с современными БД
+ простое программирование	+ возможность добавлять новые виды БД	+ возможность добавлять новые виды БД
– невозможность работать с новыми типами БД	– зависимость от драйверов, установленных в системе	– зависимость от пакета Microsoft Net Framework
– невозможность обновлять список поддерживаемых БД	– более сложное программирование	– более сложное программирование

Динамические запросы и запросы, выполненные на стороне сервера можно создавать только в технологиях RDC и ADO.Net.

10.2.3 Интерфейс информационных систем

В системах, построенных по технологии клиент-сервер существует два вида интерфейса:

- интерфейс, реализуемый при помощи клиентского приложения;
- web-интерфейс.

Интерфейс, реализуемый при помощи клиентского приложения – это компьютерная программа, устанавливаемая на клиентские компьютеры, предназначенная для работы с файлами данных через сеть. Основными элементами клиентских приложений являются формы (окно программы) и отчеты.

Элементы управления на форме называется объектами. Каждый объект обладает своим набором свойств, событий и методов.

В БД все объекты форм делятся на два класса:

– объекты управления – объекты, осуществляющие управление БД (например, кнопка или выпадающий список);

– объекты для отображения информации – элементы, отображающие содержимое таблиц, запросов или фильтров, позволяющие добавлять изменять и удалять информацию, и проводить ее анализ.

Все формы в клиентском приложении делятся на три группы:

1) формы для работы с данными – формы, содержащие как объекты управления, так и объекты просмотра данных. Такие формы предназначены для отображения, изменения, удаления и анализа данных;

2) кнопочные формы – формы, содержащие только объекты управления, предназначаются для открытия всех других форм;

3) информационные и служебные формы – формы, содержащие только элементы управления, предназначены для отображения служебной информации (справки), несвязанной с таблицами, запросами и фильтрами, либо для выполнения служебных операций, не связанных с данными (например, форма с калькулятором).

Существует два вида дизайна форм:

– ленточные формы – формы, выводящие информацию по одной записи;

– табличные формы – формы, выводящие информацию в виде таблицы.

Отчет – это объект базы данных, который используется для вывода на экран, в печать или файл структурированной информации. Отчеты позволяют извлечь из таблиц или запросов базы данных необходимую информацию и представить ее в виде удобном для восприятия. Отчеты содержит заголовок, область данных, верхний и нижний колонтитулы, примечание и разбит на страницы.

Основой web-интерфейса являются страницы (файл с расширенным htm или html). Работа со страницами осуществляется с помощью программы – браузера. Изначально страницы находятся на сервере, пользователь сначала загружает их на свой компьютер с сервера, а затем с помощью страниц пользователь работает с файлом данных.

Объекты связи используются только в клиентском интерфейсе. В web-интерфейса функции объекта связи выполняет сервер. В web-интерфейсе отсутствуют отчеты, их роль выполняют сами страницы.

10.3 Создание и настройка виртуальной машины

Под понятием виртуальная машина (Virtual Machine) понимают программную или аппаратную систему, которая эмулирует аппаратное обеспечение некой платформы (гостевая платформа), исполняющая программы для гостевой платформы средствами хост-платформы. Виртуальная машина может виртуализировать некую платформу, создавая на ней независимые, изолированные среды для работы операционных систем и программ.

Таким образом, виртуальная машина предоставляет возможность на одном реальном, физическом компьютере, создавать несколько виртуальных компьютеров, устанавливать на них различные операционные системы, программы и пр.

Технология виртуализации пришла из серверной инфраструктуры, где виртуальные машины используются с целью создания максимальной загрузки сервера и уменьшения простоев оборудования.

Виртуальные машины используют **для решения следующих задач:**

- оптимизация использования серверных ресурсов;
- информационная защита, а также ограничение возможностей некоторых программ;
- исследования новой компьютерной архитектуры или программного обеспечения;
- эмуляция различных компьютерных архитектур;
- создание вредоносного кода (предоставляющего, например, удаленный контроль над виртуальной машиной злоумышленнику).
- моделирование компьютерных сетей;

– тестирование и отладка программного обеспечения.

Одной из самых популярных программ виртуализации является VirtualBox.

VirtualBox (Oracle VM VirtualBox) – бесплатный программный продукт виртуализации для операционных систем Microsoft Windows, Linux, FreeBSD, Mac OS X, Solaris/OpenSolaris, ReactOS, DOS и других. Свободно распространяется под универсальной общественной лицензией GNU. При помощи общих сетевых ресурсов VirtualBox позволяет осуществлять обмена файлами между «внешней» и «внутренней» операционными системами.

Перед тем, как начать установку, необходимо выделить новой системе некоторые количество системных ресурсов – максимально доступное количество оперативной памяти и ядер процессора, а также назначить максимальный уровень их загрузки. Также для настройки доступны более тонкие параметры аппаратной конфигурации.

10.3.1 Установка VirtualBox

Прежде всего, на официальном сайте https://www.virtualbox.org необходимо загрузить установщик VirtualBox, активировав опцию «Download VirtualBox 5.1» (см. рисунок 10.1).

Рисунок 10.1 – Официальная страница для загрузки установщика VirtualBox

В следующем окне необходимо выбрать соответствующую операционную систему для VirtualBox. В представленном ниже примере выбран установщик для Windows «VirtualBox 5.1 for Windows hosts» (см. рисунок 10.2).

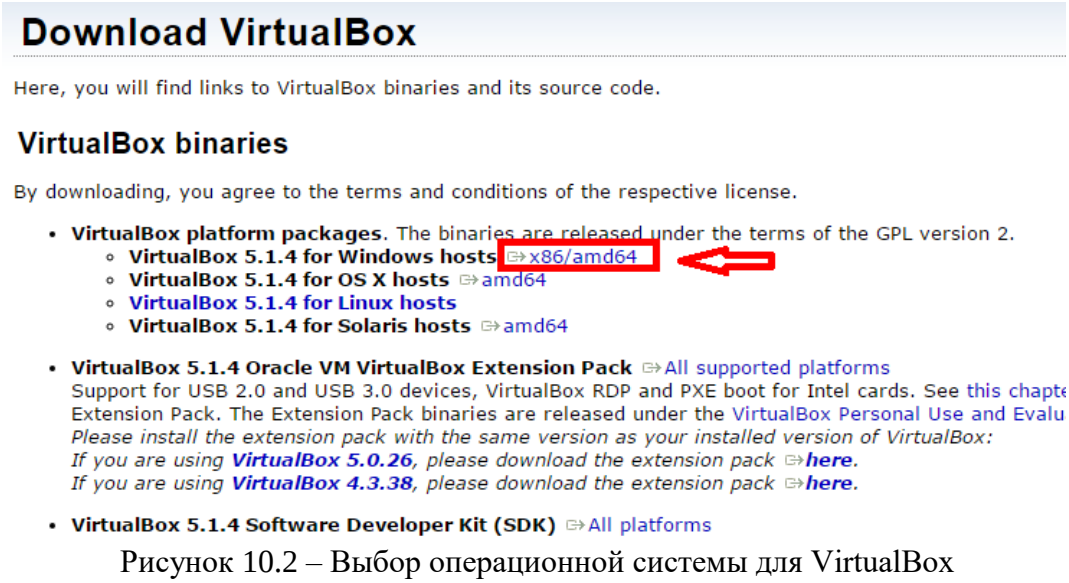


Рисунок 10.2 – Выбор операционной системы для VirtualBox

После полной загрузки установщика можно преступить к установке VirtualBox. Никаких дополнительных настроек во время установки VirtualBox производить не надо, необходимо несколько раз активировать опцию «Next», утвердительно ответить на предупреждение о временном отключении компьютера от сети и запустить процесс установки.
По завершении установки необходимо перезагрузить компьютер.

10.3.2 Создание виртуальной машины

Для создания виртуальной машины в VirtualBox необходимо активировать опцию «Создать» (см. рисунок 10.3).

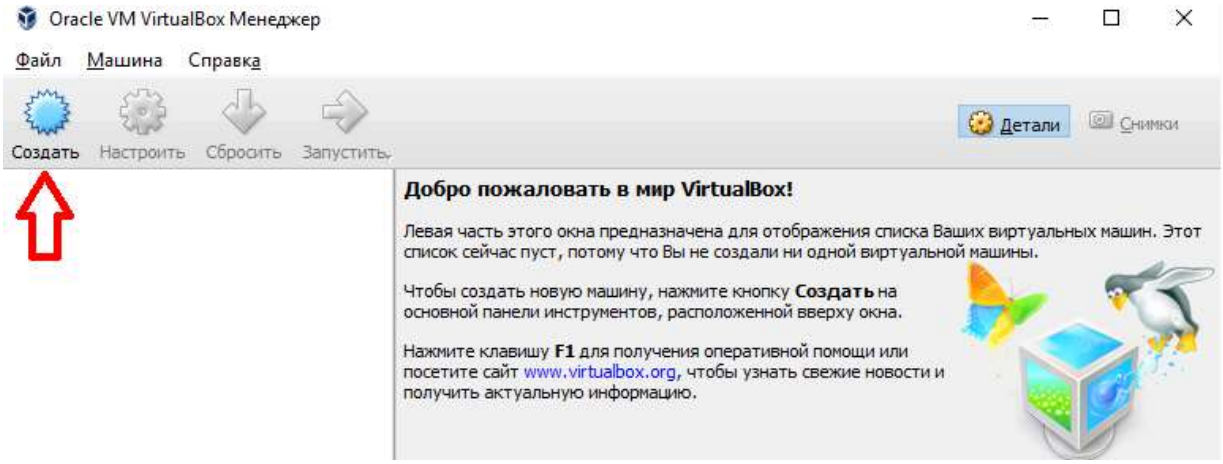


Рисунок 10.3 – Создание новой виртуальной машины

В открывшемся окне в представленных полях необходимо вписать следующие данные (см. рисунок 10.4):

- имя – вписать название создаваемой виртуальной машины;
- тип – выбрать Linux;
- версия – выбрать Ubuntu (64-bit);

Затем необходимо активировать опцию «Next».

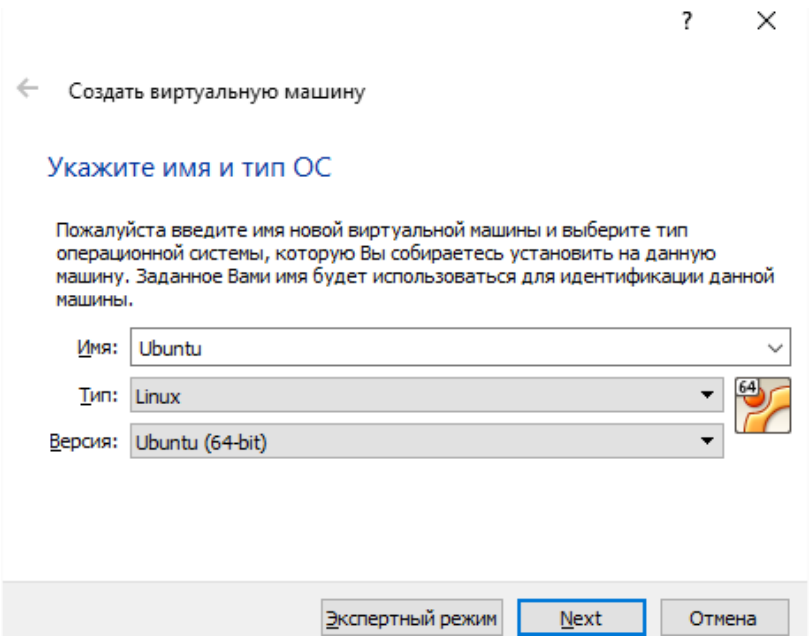


Рисунок 10.4 – Выбор типа виртальнй машины

В следующем окне необходимо установить выделяемый объем оперативной памяти для виртуальной машины. В представленном ниже примере (см. рисунок 10.5). Активировать опцию «Next».

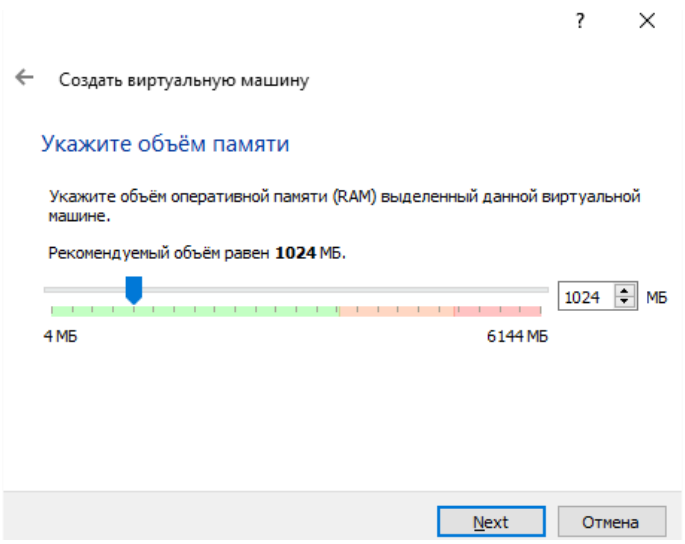


Рисунок 10.5 – Установка объема оперативной памяти для виртуальной машины

В следующем окне из предложенного списка необходимо выбрать вариант «Создать новый виртуальный жесткий диск» и активировать опцию «Создать» (см. рисунок 10.6).

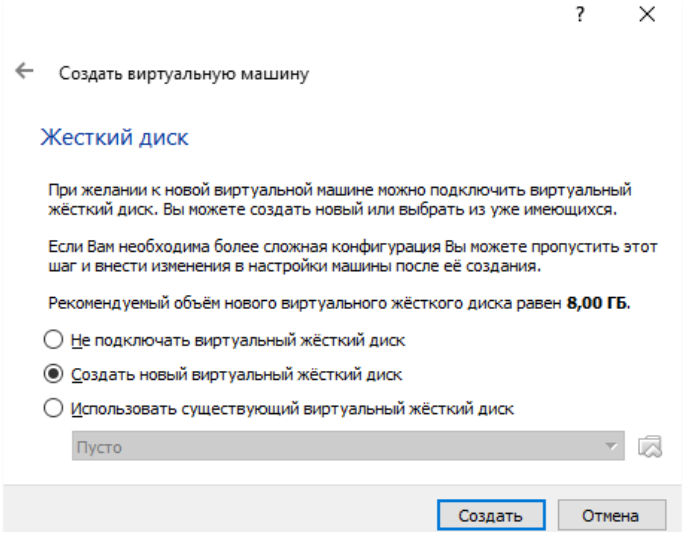


Рисунок 10.6 – Создание нового виртуального диска

В следующем окне необходимо указать тип файла, определяющего формат жесткого диска – VDI, которые выбран по умолчанию и активировать опцию «Next» (см. рисунок 10.7).

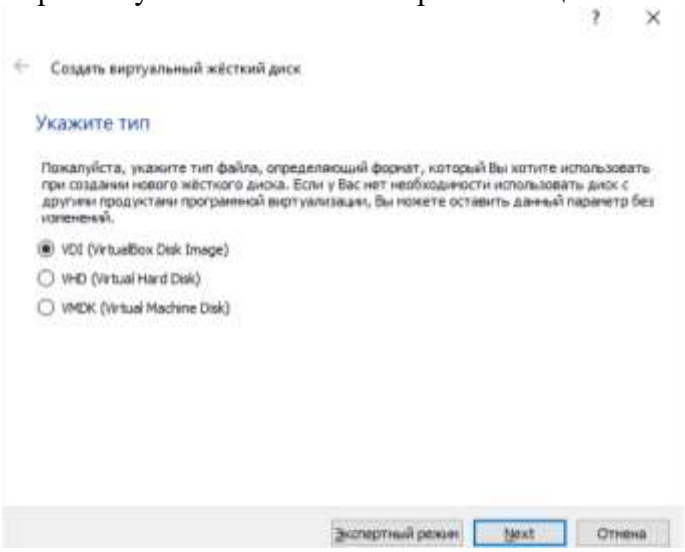


Рисунок 10.7 – Выбор типа файла, определяющего формат жесткого диска

В следующем окне необходимо выбрать вариант хранения данных «Динамический виртуальный жесткий диск» и активировать опцию «Next» (см. рисунок 10.8).

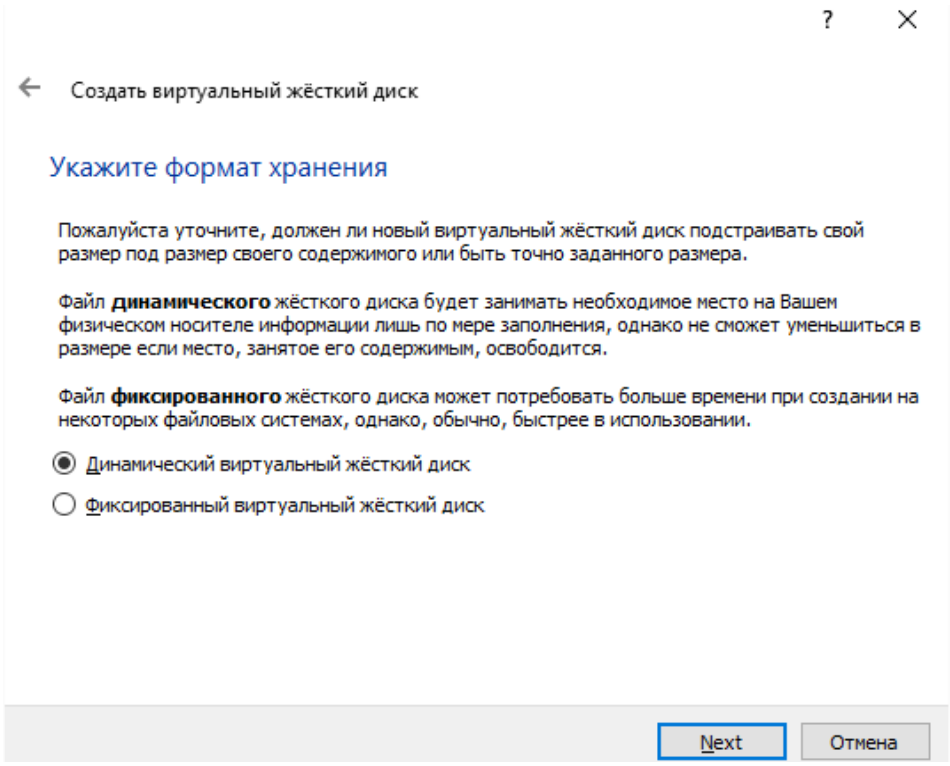


Рисунок 10.8 – Выбор формата хранения данных на виртуальном жестком диске

В следующем окне необходимо указать объем виртуального жесткого диска. Система рекомендует 8ГБ, можно оставить без изменений и активировать опцию «Создать» (см. рисунок 10.9).

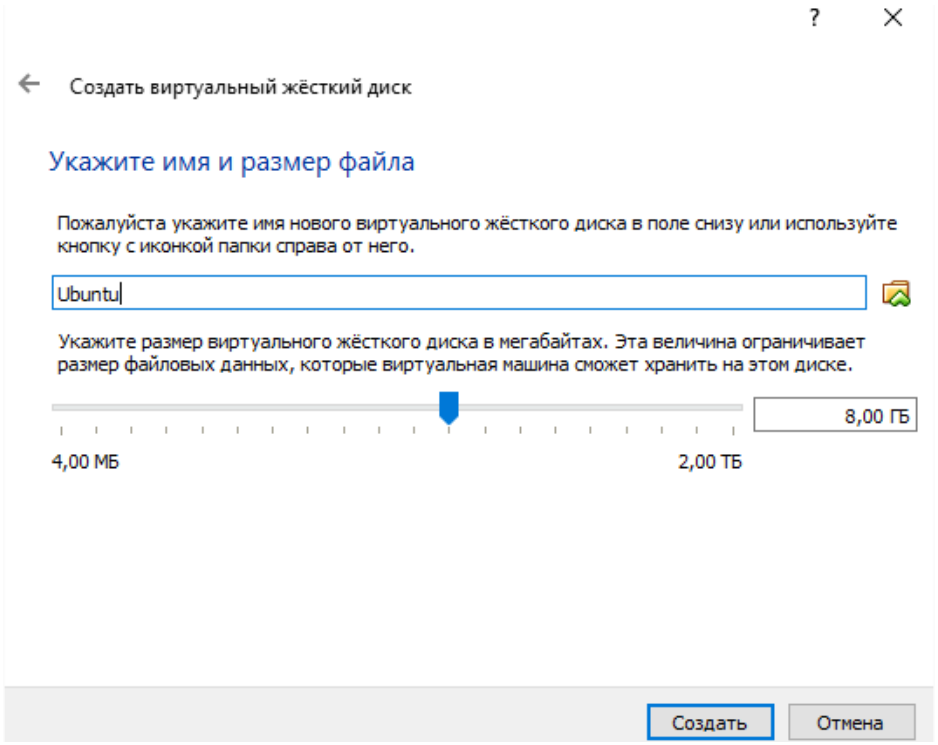


Рисунок 10.9 – Установка объем виртуального жесткого диска

Теперь виртуальная машина создана и ее можно увидеть в списке виртуальных машин (см. рисунок 4.10).

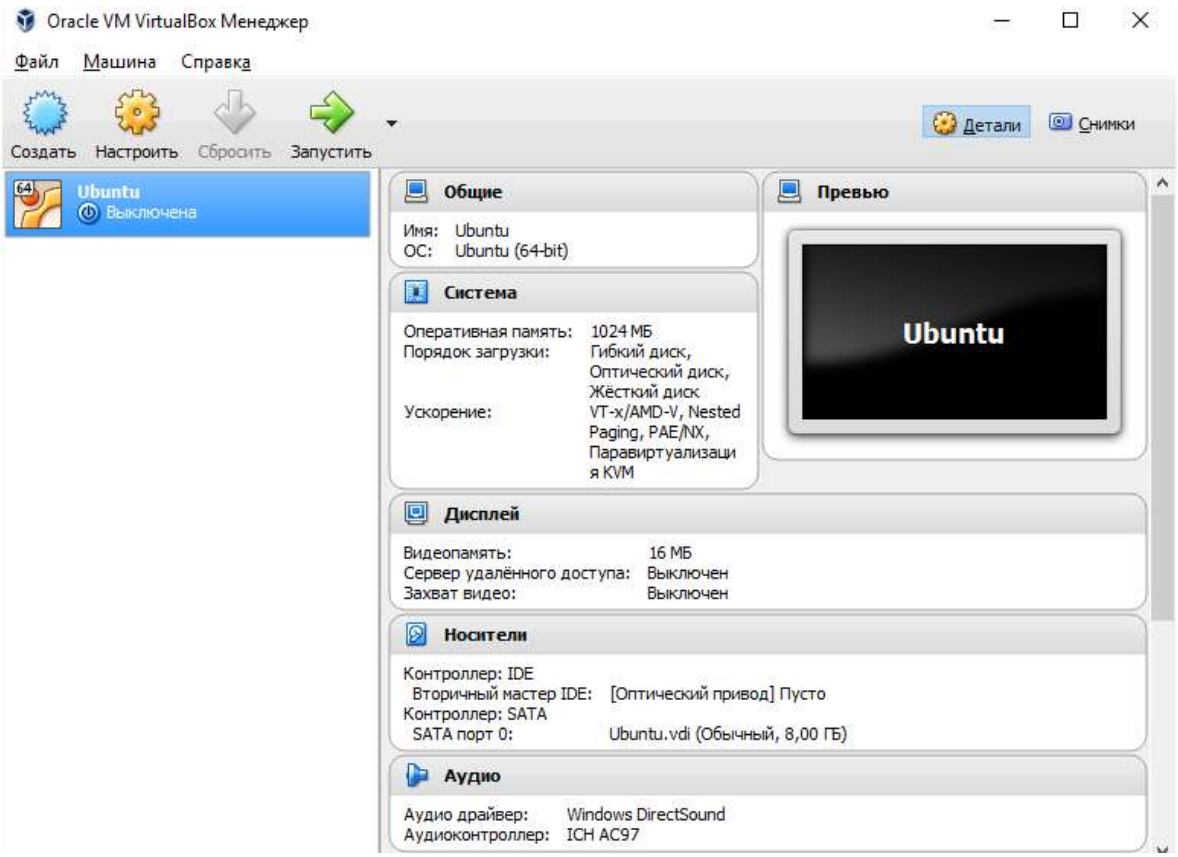


Рисунок 10.10 – Список виртуальных машин, созданных в VirtualBox

Теперь можно приступать к установке Ubuntu Desktop на созданную виртуальную машину.

10.3.3 Установка операционной системы Ubuntu Desktop на виртуальную машину

Прежде чем приступить к установке Ubuntu, необходимо скачать сам дистрибутив. Для этого надо зайти на официальный сайт Ubuntu http://www.ubuntu.com и в меню «Download» выбрать пункт «Desktop» и активировать опцию «Alternative downloads and torrents» (см. рисунок 10.11). На момент написания методических указаний актуальной была версия 16.04.1.

Рисунок 10.11 – Страница официального сайта для загрузки установщика Ubuntu

В появившемся окне необходимо выбрать раздел «Ubuntu 16.04.1 Desktop (32 bit)» и активировать загрузку torrent-файла установщика (подходящего под тип процессора реальной машины), с помощью которого затем можно загрузить сам установщик операционной системы Ubuntu (см. рисунок 10.12).

Рисунок 10.12 – Выбор версии операционной системы Ubuntu

После закачки установщика операционной системы можно приступать к установке Ubuntu на виртуальную машину.

Для этого необходимо поместить только что скачанный образ ISO в привод виртуальной машины. В разделе «Носители» в опции «Оптический привод» необходимо выбрать только что скачанный образ дистрибутива Ubuntu Desktop (см. рисунок 10.13) и запустить виртуальную машину, активировав опцию «Запустить».

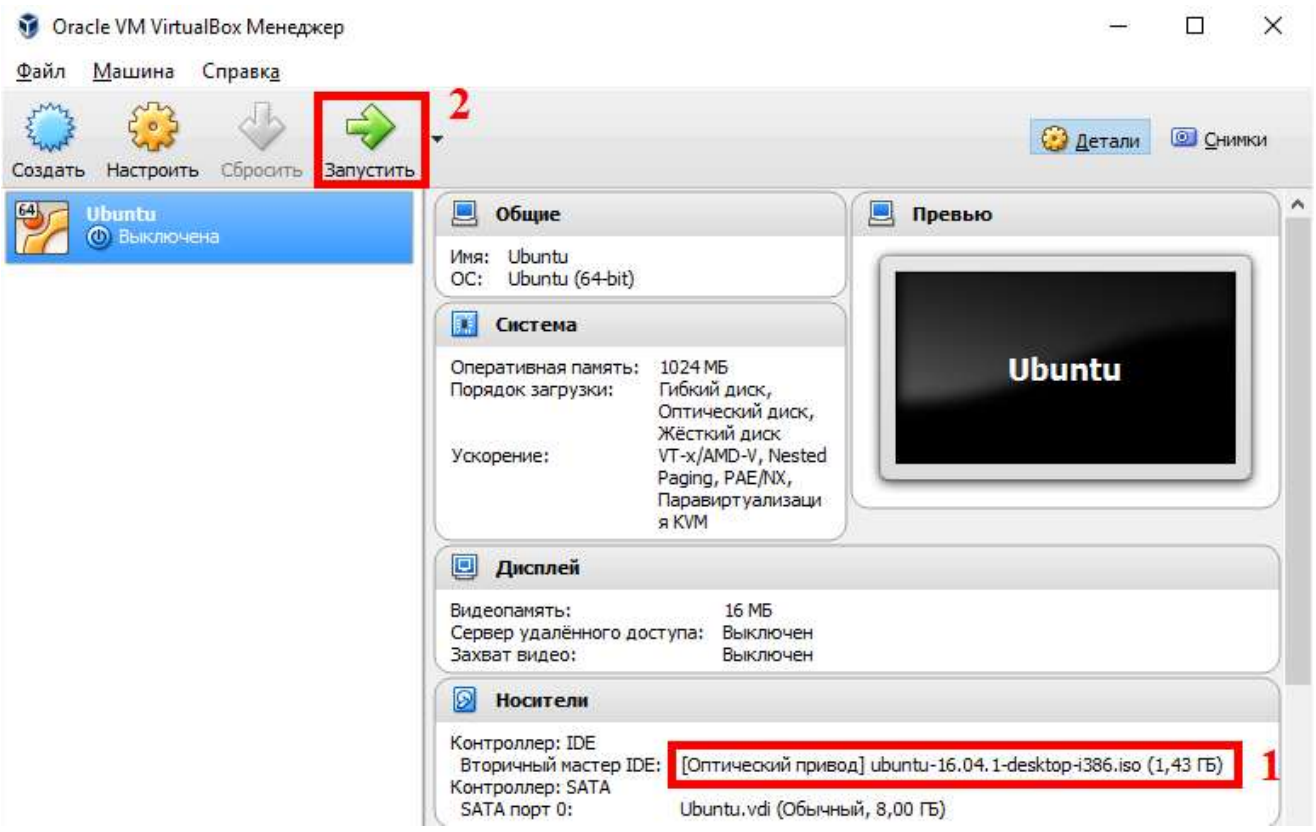
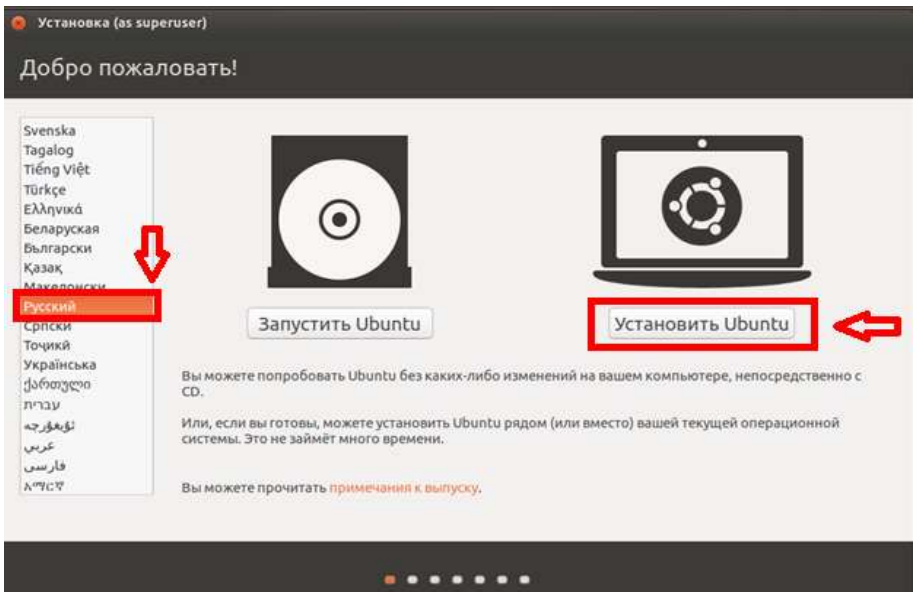


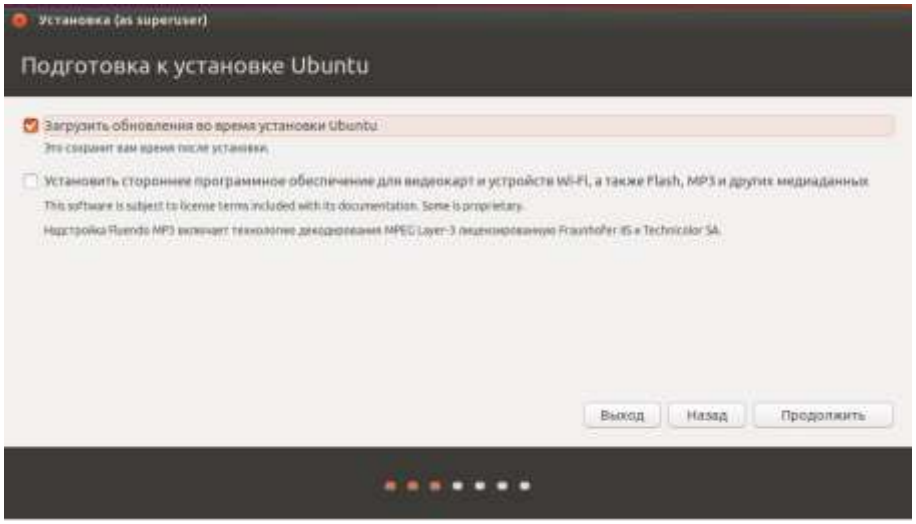
Рисунок 10.13 – Выбор образа для установки операционной системы

Автоматически начнется установка операционной системы Ubuntu 16.04.1 Desktop (32 bit) на виртуальную машину.

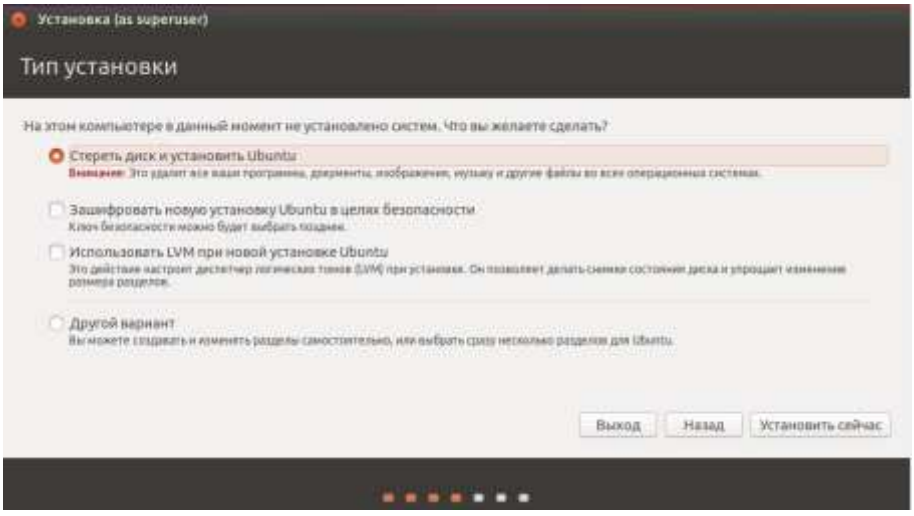
Шаг 1. Необходимо выбрать язык операционной системы и активировать опцию «Установить Ubuntu».

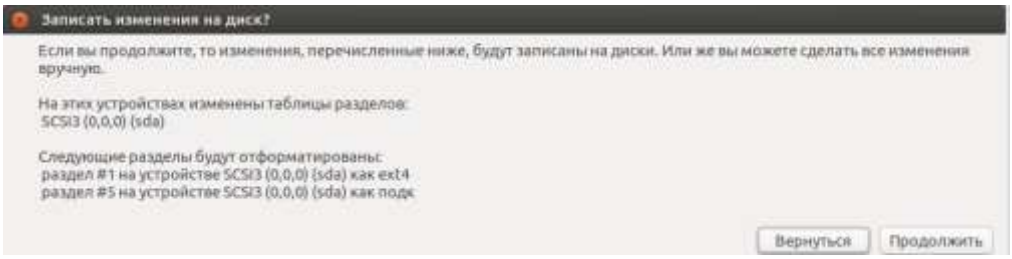


Шаг 2. Необходимо активировать опцию «Загрузить обновления во время установки Ubuntu», но для того, чтобы обновления загрузились необходимо установить соединение с Интернет и активировать опцию «Продолжить».

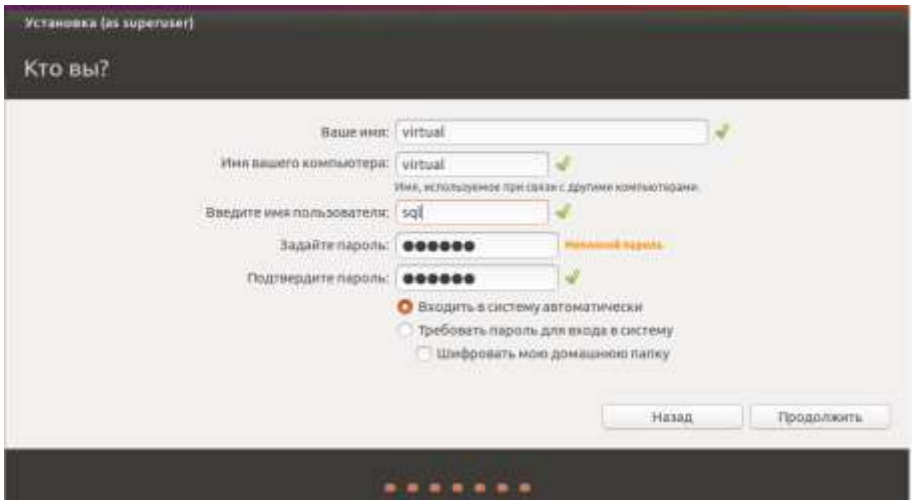


Шаг 3. На следующем шаге установки Ubuntu необходимо выбрать пункт «Стереть диск и установить Ubuntu», активировать опцию «Установить сейчас» и принять появившееся предупреждение.

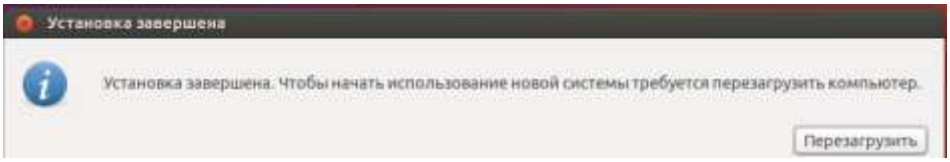




Шаг 4. На следующем шаге установки Ubuntu необходимо задать имя компьютера и учетные данные пользователя: логин, пароль. Также необходимо активировать опцию «Входить в систему автоматически» для того, чтобы при каждом запуске виртуальной машины не вводить пароль пользователя. После активации опции «Продолжить» начнется установка операционной системы Ubuntu на выбранную виртуальную машину.



По завершении установки операционной системы будет выведено соответствующее сообщение с предложением перезапустить компьютер.



После перезапуска компьютера и активации опции «Enter», на виртуальной машине отобразится рабочий стол операционной системы Ubuntu.

10.3.4 Установка MySQL и MySQL Workbench на Ubuntu

Установка и настройка программного обеспечения в Ubuntu производится в консоли. Обязательным условием является наличие Интернет-соединения. Вызов консоли Ubuntu осуществляется комбинацией клавиш Ctrl+Alt+T.

Для установки MySQL-сервера и клиента, в консоли Ubuntu необходимо выполнить команду:

```
sudo apt-get install mysql-server mysql-client
```

В процессе установки, в консоли потребуется дважды ввести пароль администратора сервера MySQL.

Для установки графического клиента Workbench для работы с MySQL в строке поиска менеджера приложений Ubuntu необходимо ввести команду «MySQL Workbench» и активировать опцию «Установить».

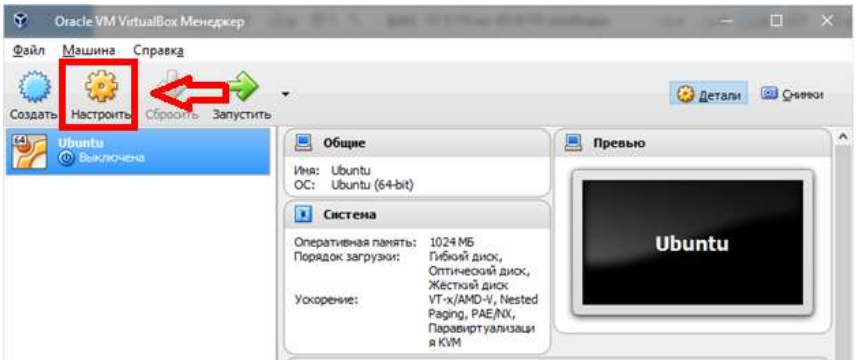
10.4 Создание и настройка сети между реальной и виртуальной машинами

10.4.1 Настройка сети между реальной и виртуальной машинами

Для создания и настройки виртуальной компьютерной сети между реальной и виртуальной машинами средствами программы VirtualBox в списке виртуальных машин необходимо выбрать ту машину, с которой необходимо будет настроить сетевое подключение, и активировать опцию «Настроить».

В левой части меню необходимо перейти в раздел «Сеть».

Каждая виртуальная машина имеет 4 условных адаптера, каждый из адаптеров имеет 5 профилей настройки.



Настройку виртуальной компьютерной сети между реальной и виртуальной машинами средствами программы VirtualBox можно реализовать двумя способами:

- используя тип соединения «Сетевой мост» – в этом случае условный сетевой адаптер подключается и работает напрямую с физическим адаптером минуя хост-машину. Данный тип работы адаптера есть смысл использовать, когда необходимо предоставить доступ к виртуальной машине другим участникам локальной физической сети;
- используя тип соединения «Виртуальный адаптер хоста» – при таком режиме работы есть возможность взаимодействия как между виртуальными машинами, а также виртуальной машиной и хостом. При использовании адаптера хоста отсутствует возможность взаимодействия с другими участниками физической локальной сети.

Настройка сетевого моста (1 способ). Для данного типа соединения необходимо выбрать тип подключения «Сетевой мост (Bridge)» и в дополнительных настройках активировать неразборчивый режим, выбрав надстройке «Разрешить все» (см. рисунок 10.14).

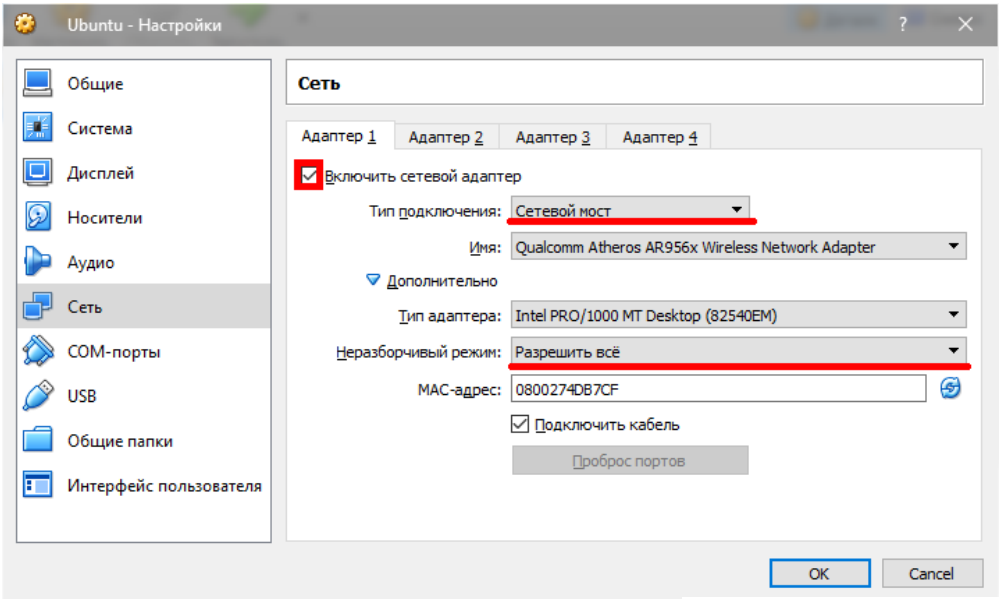


Рисунок 10.14 – Настройка сетевого подключения между между реальной и виртуальной машинами

Используя этот тип соединения, виртуальная машина ничем не отличается от хост-машины для других участников сети, сетевой адаптер при такой настройке служит мостом между виртуальной и физической сетью. Таким образом, данный тип работы адаптера используется, когда необходимо предоставить доступ к виртуальной машине другим участникам локальной физической сети.

Условный сетевой адаптер подключается и работает напрямую с физическим адаптером минуя хост-машину.

Если компьютер имеет несколько сетевых интерфейсов, то в поле «Имя» необходимо указать через какой из них будет осуществляться взаимодействие.

Настройка виртуального адаптера хоста (2 способ). Для данного типа соединения необходимо выбрать тип подключения «Виртуальный адаптер хоста» и в дополнительных настройках активировать неразборчивый режим, выбрав надстройке «Разрешить все» (см. рисунок 10.15).

В этом случае используется специальное устройство – VirtualBox Host-Only Ethernet Adapter, которое создает подсети и назначает IP-адреса гостевым операционным системам.

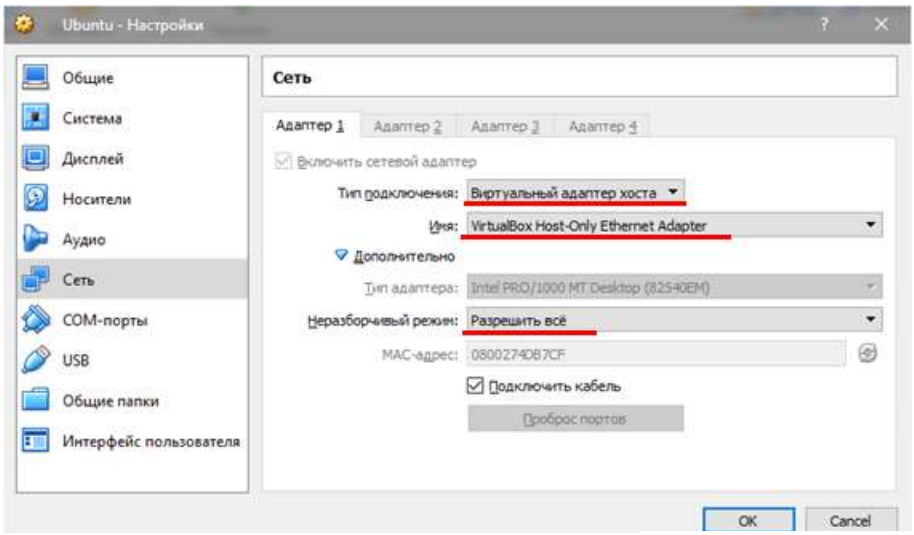


Рисунок 10.15 – Настройка сетевого подключения между между реальной и виртуальной машинами

10.4.2 Установка статического IP-адреса

По умолчанию виртуальной машине автоматически присваивается динамический IP-адрес, который при каждом новом запуске может быть разным. Поэтому, для того, чтобы обеспечить постоянное стабильное подключение к серверу MySQL, тип IP-адреса, выделяемого виртуальной машине, необходимо изменить с динамического на статический.

Для этого в операционной системе Ubuntu на виртуальной машине необходимо зайти в меню «Настройки» и выбрать раздел «Сеть» (см. рисунок 10.16).

Рисунок 10.16 – Вход в меню сетевых настроек виртуальной машины

В появившемся окне необходим запомнить текущий IP-адреса виртуальной машины (на рисунке 10.17 это адрес IPv4 – 192.168.0.105) и шлюз по умолчанию, если указан (на рисунке 10.17 это маршрут по умолчанию – 192.168.0.1). Затем необходимо выбрать тип соединения – «Проводное» и активировать опцию «Параметры…» (см. рисунок 10.17).

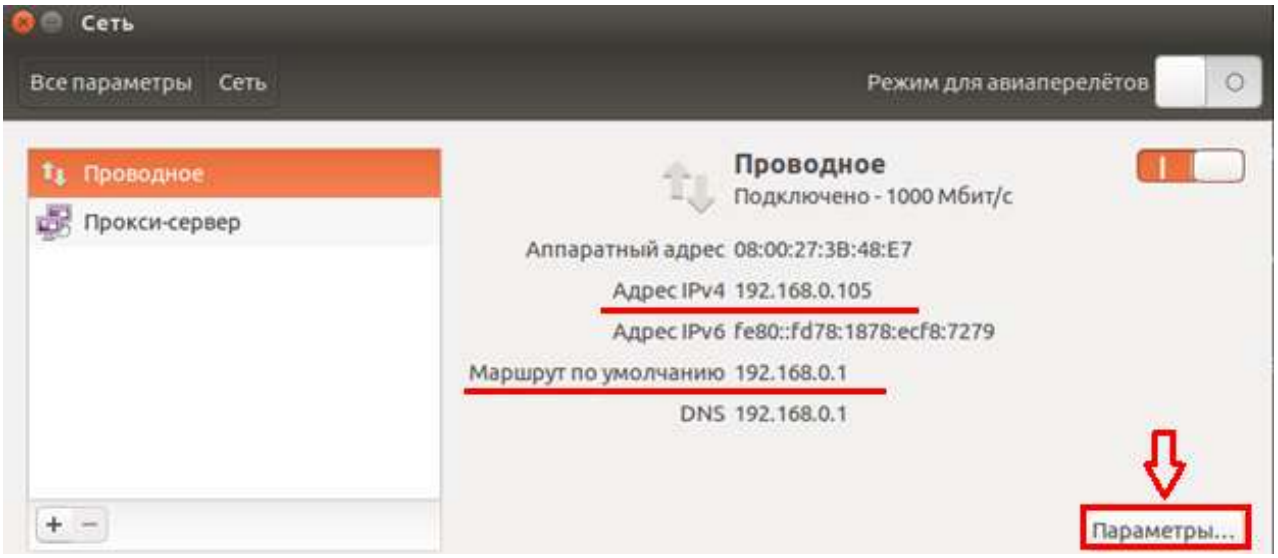


Рисунок 10.17 – Меню сетевых настроек виртуальной машины

Затем, в окне изменения настроек проводного соединения необходимо выбрать раздел «Параметры IPv4» и в меню «Способ настройки» активировать опцию «Вручную» вместо предложенной «Автоматически (DHCP)» (см. рисунок 10.18).

Затем необходимо сделать динамически выделенный виртуальной машине IP-адрес статическим. Для этого необходимо активировать опцию «Добавить» и в разделе «Адрес» в столбце «Адрес» вручную прописать IP-адрес, который был присвоен динамически (см. рисунок 10.17, поле «Адрес IPv4»), в столбце «Маска подсети» – указать стандартное значение маски – 255.255.255.0, а в столбце «Шлюз» – значение маршрута по умолчанию (см. рисунок 10.17), и затем активировать опцию «Сохранить» (см. рисунок 10.18).

Настройка статического IP-адреса для виртуальной машины завершена.

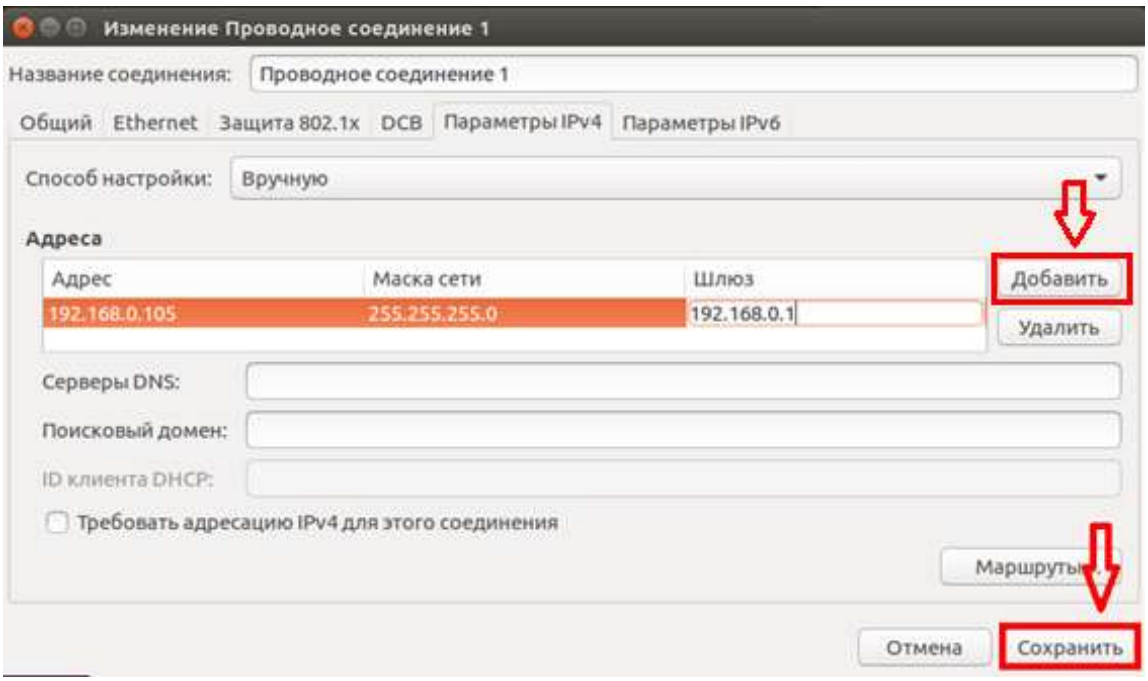


Рисунок 10.18 – Изменение настроек проводного соединения

10.4.3 Настройка удаленного соединения с сервером MySQL

По умолчанию сервер MySQL принимает соединения только с локальной машины. Для того, чтобы разрешить подключаться к нему с других машин, необходимо открыть конфигурационный файл **mysqld.cnf**, с возможностью редактирования, с помощью следующей команды консоли Ubuntu:

```
sudo gedit /etc/mysql/mysql.conf.d/mysqld.cnf
```

И в появившемся окне найти и замените строку:

```
bind-address            = 127.0.0.1
```

на

```
bind-address            = 0.0.0.0
```

Данная команда позволяет настроить MySQL на ожидание подключений от компьютеров в сети, если в параметре «bind-address» указать IP-адрес 0.0.0.0, то подключение будет разрешено с любого компьютера.

После сохранения произведенных изменений к серверу MySQL можно подключаться извне.

Осталось создать пользователей, которым разрешено удаленное подключение. Для этого необходимо запустить консоль MySQL командой:

```
mysql –u root –p
```

И создать пользователя командой:

```
GRANT ALL PRIVILEGES ON *.* TO username@"remote_addr"
        IDENTIFIED BY 'password' WITH GRANT OPTION;
```

Вместо значения «**remote_addr**» необходимо указать адрес хоста с которого планируется подключение или можно указать значение «**%**» тогда подключение будет разрешено с любого адреса, в поле «**password**» необходимо указать пароль пользователя. Например, командой, представленной ниже, можно добавить пользователя с именем **root** и паролем **12345** для удаленного подключения к серверу MySQL:

```
GRANT ALL PRIVILEGES ON *.* TO root@"%"
        IDENTIFIED BY '12345' WITH GRANT OPTION;
```

Затем необходимо выполнить команду для обновления таблиц MySQL, в которых хранится информация о пользователях:

```
mysqladmin -u root -p flush-privileges;
```

10.5 Синхронизация структуры данных

10.5.1 Создание удаленного подключения к серверу MySQL

Теперь можно настроить удаленное подключение к серверу MySQL с клиентской (реальной) машины.

Для этого на стартовом окне приложения MySQL Workbench в разделе подключений к серверу MySQL «MySQL Connections» необходимо добавить новое подключение, активировав опцию  .

В появившемся окне необходимо указать IP-адрес машины, на которой развернут MySQL-сервер (например, 192.168.0.105), имя пользователя, который подключается, и пароль (см. рисунок 10.19).

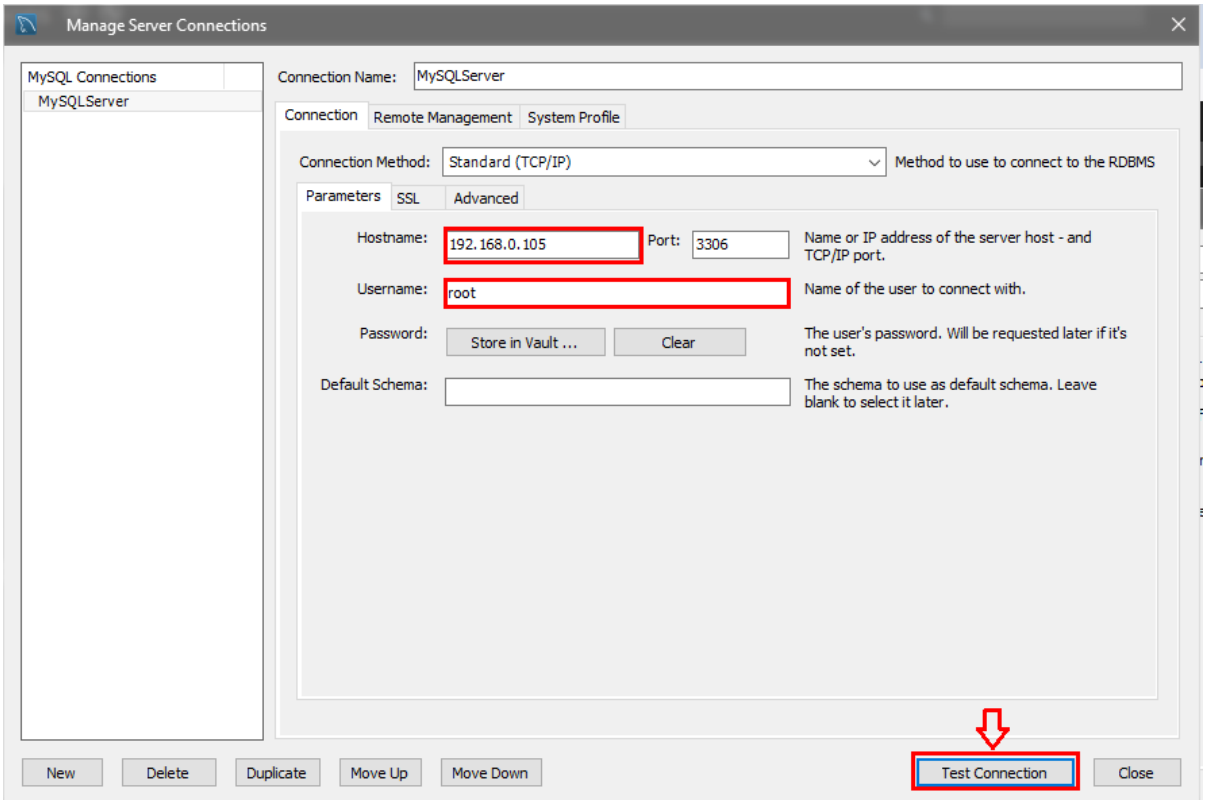


Рисунок 10.19 – Настройка подключения к удаленному серверу MySQL

Активировав опцию «Test connection» необходимо проверить статус подключения. Если все сделано правильно, появится сообщение об успешном установлении соединения с сервером MySQL (см. рисунок 10.20).

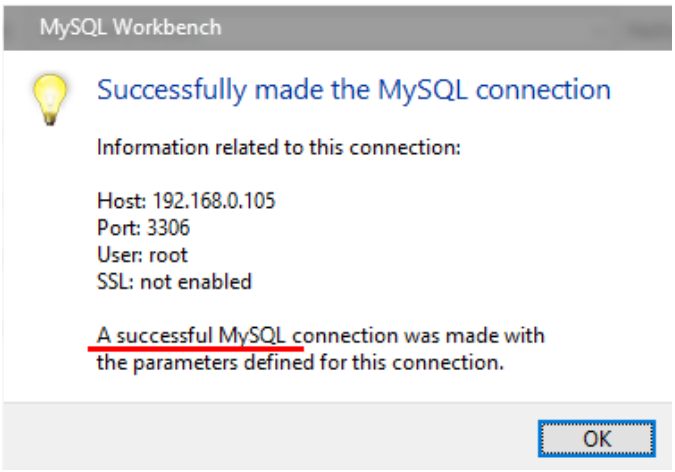


Рисунок 10.20 – Проверка подключения к удаленному серверу MySQL

10.5.2 Экспорт модели MySQL Workbench на SQL-сервер

Самый быстрый путь для того, чтобы схема данных из MySQL Workbench попала на сервер – **создание SQL дампа mwb-модели.** Для этого не потребуется создавать удаленное подключение в программе, однако этот способ хорош лишь в случае, если требуется однократная заливка структуры и базовых данных на сервер. Дальнейшая поддержка, обновление и синхронизация модели данных в этом случае будет весьма проблематична.

Для экспорта существующей модели MySQL Workbench на SQL-сервер необходимо открыть модель и выбрать пункт меню *«File → Export → Forward Engineer SQL CREATE Script...». На рисунке 10.21 представлено окно настроек экспорта.*

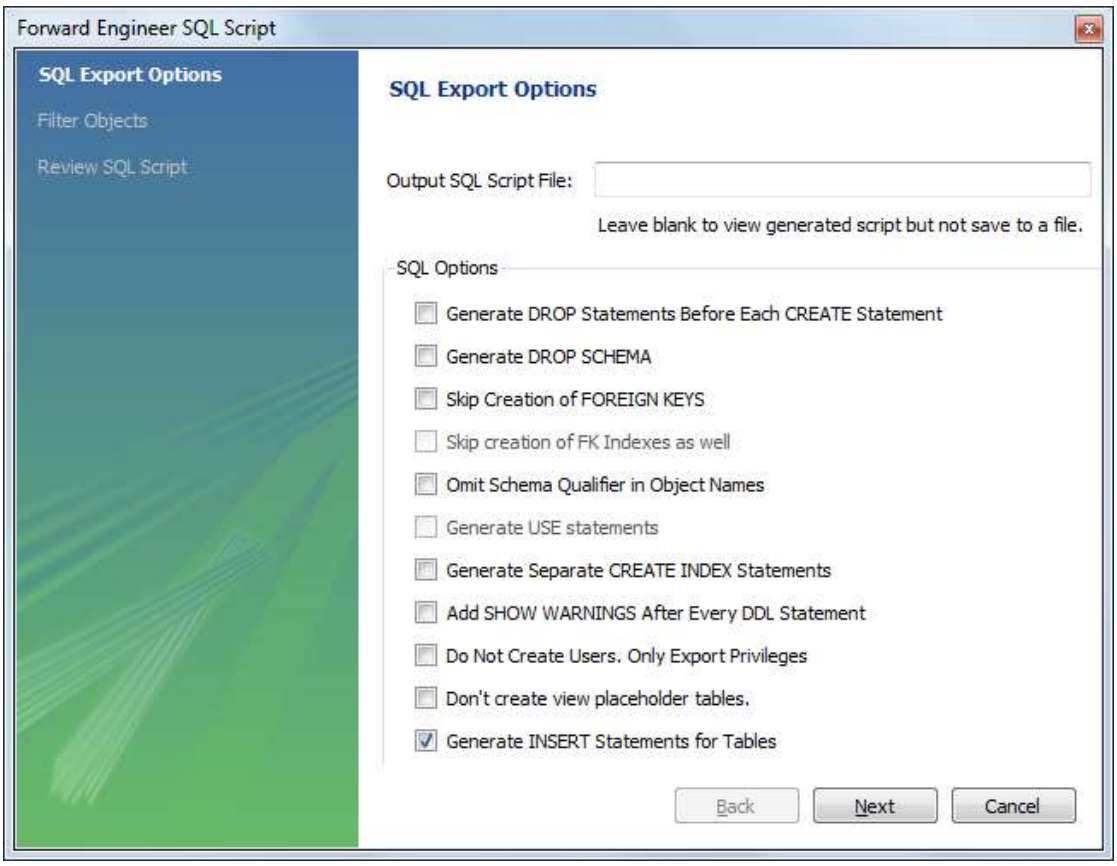


Рисунок 10.21 – Окно настроек экспорта модели MySQL Workbench на сервер

Если требуется записать SQL-скрипт в файл, необходимо указать путь до файла в поле «Output SQL Script File» (если оставить поле пустым, SQL-скрипт можно будет скопировать на последнем шаге в буфер обмена).

Настройки стандартные, чтобы понять их суть, достаточно перевести их названия. Если активировать опцию «Generate INSERT Statements for Tables» в дамп будут включены базовые данные, располагающиеся во вкладке «Inserts» интерфейса редактирования таблиц модели. После активации опции «Next» появится список объектов, которые можно экспортировать. Для экспорта таблиц необходимо выбрать опцию «Export MySQL Table Objects», а чтобы экспортировать их выборочно, можно дополнительно активировать опцию «Show Filter» и выбираем нужные таблицы (см. рисунок 10.22).

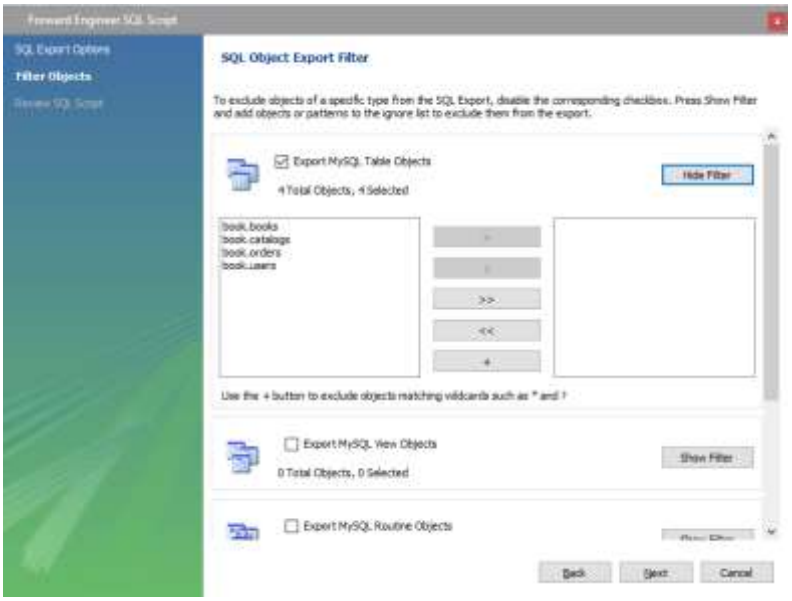


Рисунок 10.22 – Выбор объектов для экспорта

Активировав опцию «Next», в следующем окне появится готовый SQL-скрипт, который, при необходимости, можно скопировать его буфер обмена или записать в файл.

10.5.3 Синхронизация структуры данных

Для синхронизации структуры базы данных и локальной модели в MySQL Workbench существует специальный инструмент. Необходимо открыть нужную модель и выбрать пункт меню «Database → Synchronize Model...», после чего необходимо выбрать сохраненное в п. 10.3.2 удаленное подключение и отредактировать его параметры.

Для подключения к базе данных необходимо активировать опцию «Next». В новом окне появится список моделей (в левой колонке) и баз данных (в правой колонке), доступные для синхронизации (см. рисунок 10.23).

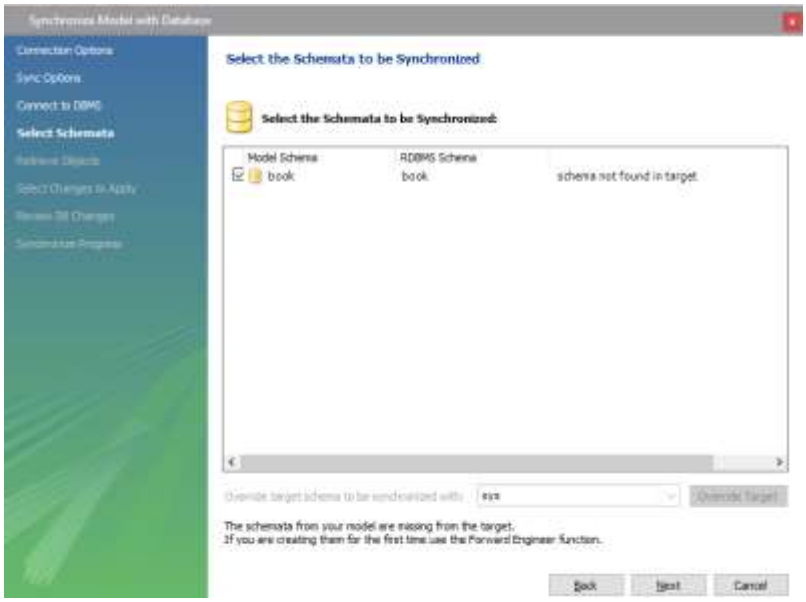


Рисунок 10.23 – Доступные модели и базы данных для синхронизации

Выбрав нужную базу и схему, необходимо снова активировать опцию *«Next»*, запустив тем самым процедуру сравнения структур удаленной базы данных и локальной модели. После завершения процедуры появится список различий между локальной схемой данных и удалённой базой (см. рисунок 10.24).

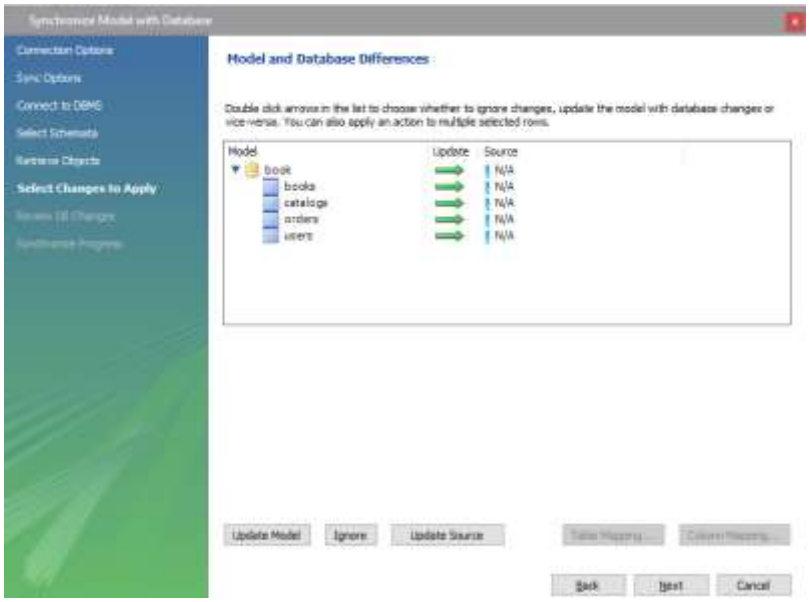


Рисунок 10.24 – Список различий между локальной и удаленной схемами данных

На данном этапе можно настроить объединение таблиц: протолкнуть изменения на сервер («Update Source»), втянуть в локальную модель конфигурацию с сервера («Update Model») или игнорировать отличия («Ignore»). Кроме того, доступен как вариант настройки для всей базы, так и отдельно для каждой таблицы. При выделении одной из таблиц и выборе способа объединения можно увидеть SQL-запросы, которые выполнятся в процессе синхронизации, а активировав опцию «Next» – увидеть полный стек этих запросов.

Просмотрев SQL-запросы, активировав опцию «Execute >», начнется выполнение синхронизации. Если все пройдет успешно, появится сообщение, представленное на рисунке 10.25.

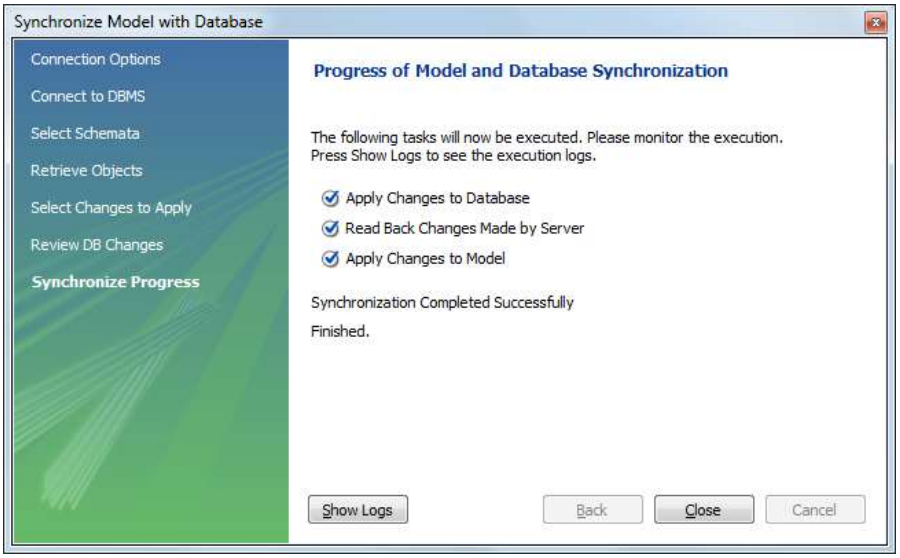


Рисунок 10.25 – Отчет об успешно выполненной синхронизации

В случае возникновения ошибок их лог отобразится в этом же диалоговом окне.

10.5.4 Выгрузка на сервер схемы и стартовых данных

Описанная выше синхронизация осуществляет лишь объединение структуры схемы данных удаленной базы и локальной модели, но никак не затрагивает стартовые данные, внесённые в модель («Inserts»). Если требуется выгрузить их, необходимо выбрать пункт меню «Database → Forward Engineer...», затем выбрать одно из сохраненных ранее подключений (или создать новое) и активировать опцию «Next». В остальном механизм выгрузки аналогичен механизму экспорта mwb-модели, описанному в п. 10.4.2. Его можно так же использовать, если требуется простая выгрузка схемы данных на сервер без синхронизации.

10.6 Задание для самостоятельного выполнения

1. Нормализовать отношения в базе данных, построенной в лабораторной работе №1, до третьей и четвертой нормальной формы.
2. Построить полную атрибутивную модель базы данных в нотации IDEF1X.
3. Создать виртуальную машину, установить на ней операционную систему Ubuntu и настроить на ней SQL-сервер.
4. Настроить удаленное подключение к виртуальной (серверной) машине с реальной (клиентской) и выгрузить на сервер разработанную в MySQL Workbench схему.
5. Разработать приложение для организации доступа к данным, хранящихся в БД, которая была разработана в лабораторных работах №№1, 2 и 3. БД необходимо создать под управлением выбранной Вами открытой СУБД. Приложение должно содержать кнопки и выпадающие списки. Каждая кнопка или каждый из элементов выпадающего списка обеспечивает просмотр одной формы с данными или выполнение одного запроса. Приложение должно обеспечивать просмотр всех данных БД и выполнение всех необходимых запросов,

удовлетворяющих требованиям технического задания (соответствующих описанию предметной области, реализованном в лабораторной работе №1).

6. Разрабатываемое программное приложение должно:

- заносить информацию в созданную базу данных;
- выполнять необходимые действия по модификации и удалению информации в базе данных, причем все операции по занесению, модификации и удалению данных должны выполняться в терминах предметной области, а не базы данных;
- поддерживать целостность базы данных, не допуская появления некорректных данных;
- выполнять все действия над базой данных в рамках транзакций;
- содержать достаточное количество данных, позволяющих показать результаты выполнения запросов;
- контролировать все вводимые данные.

7. Отразить в отчете к лабораторной работе:

- постановку задачи, решаемой разработанным программным приложением;
- описать технологии, используемые для доступа к базе данных;
- обосновать необходимость использования выбранной СУБД и языка программирования для реализации поставленных задач;
- указать последовательность действий (использованные команды, подключаемые библиотеки) для создания в приложении связи с внешней базой данных;
- указать команды, используемые для создания и завершения сеанса между клиентом и сервером для передачи запросов в СУБД;
- указать команды, используемые для выполнения SQL-запросов и обработка их результатов;
- привести руководство пользователя разработанным программным приложением, содержащее описание интерфейсов всех функций приложения.

10.7 Порядок защиты работы

1. Показать работу программы.
2. Объяснить:
   - принципы методологии IDEF1X;
   - нормализация до 4НФ и 3НФ;
   - принципами организации архитектуры «клиент–сервер» в системах баз данных;
   - каково назначение сервера баз данных;
   - функции клиентского приложения баз данных.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Амблер Скотт, Садаладж Прамодкумар Дж., Птицын Константин. Рефакторинг баз данных. Эволюционное проектирование – М.: ООО «Вильямс», 2007 – 368 с.

2. Астахова И.Ф., Мельников В.М., Толстобров А.П. СУБД: язык SQL в примерах и задачах – М.: «Физматиздат», 2009 – 168 с.

3. Борри Хелен. Firebird. Руководство разработчика баз данных – издательство «БХВ-Петербург», 2007 – 1104 с.

4. Грофф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. SQL. Полное руководство – М.: ООО «Вильямс», 2016 – 960 с.

5. Зрюмов Е.А., Зрюмова А.Г. Базы данных для инженеров – АлтГТУ, 2010 – 130 с.

6. Карвин Билл. Программирование баз данных SQL. Типичные ошибки и их устранение – издательство «Рид Групп», 2012 – 336 с.

7. Кригель Алекс, Трухнов Борис. SQL. Библия пользователя, 2-е издание: Пер. с англ. – М.: ООО «Вильямс», 2010 – 752 с.

8. Кузин А В., Левонисова С.В. Базы данных – издательство «Academia», 2012 – 320 с.

9. Маркин А.В. Построение запросов и программирование на SQL– М.: «Диалог-МИФИ», 2014 – 384 с.

10. Нестеров С. А. Базы данных: учебник и практикум для академического – М.: Издательство Юрайт, 2018 – 230 с.

11. Новиков Б., Горшкова Е. Основы технологий баз данных. Учебное пособие – ДМК Пресс, 2019 – 240 с.

12. Ульман Джеффри Д., Уидом Дженнифер, Гарсиа-Молина Гектор. Системы баз данных. Полный курс – М.: ООО «Вильямс», 2004 – 1088 с.

13. Фиайли К. SQL: Пер. с англ. – М.: ДМК Пресс, 2013 – 456 с.

ПРИЛОЖЕНИЕ А
Варианты заданий к лабораторной работе №1

Вариант 1

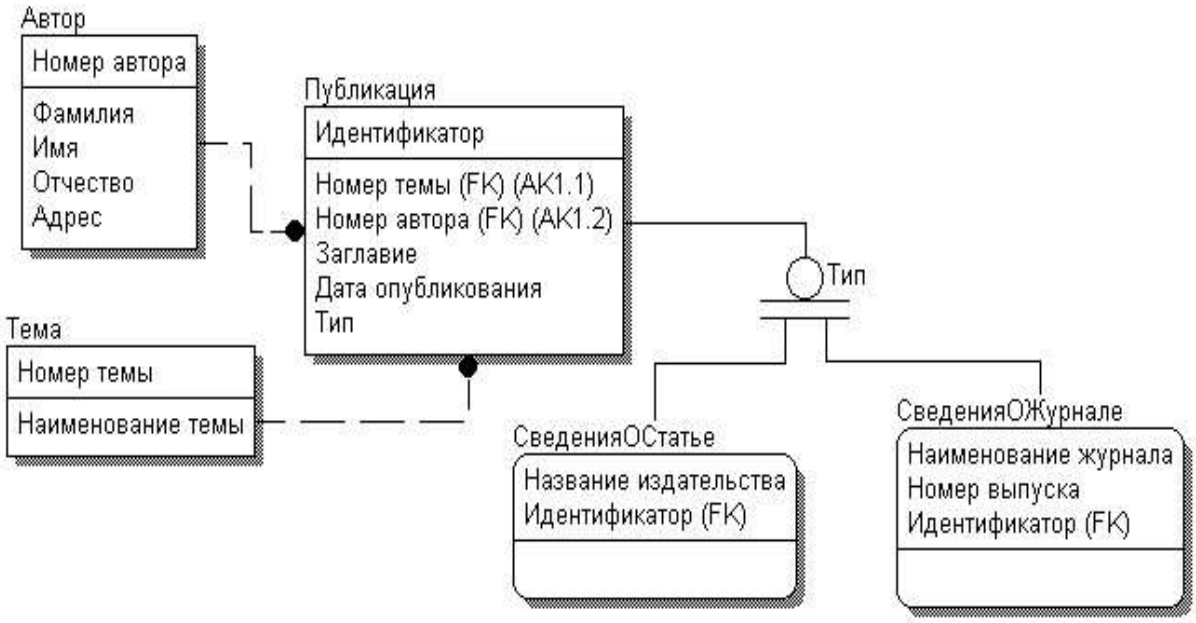


Рисунок А.1 – Структура системы, которая содержит информацию о публикациях по ряду выбранных тем

Вариант 2

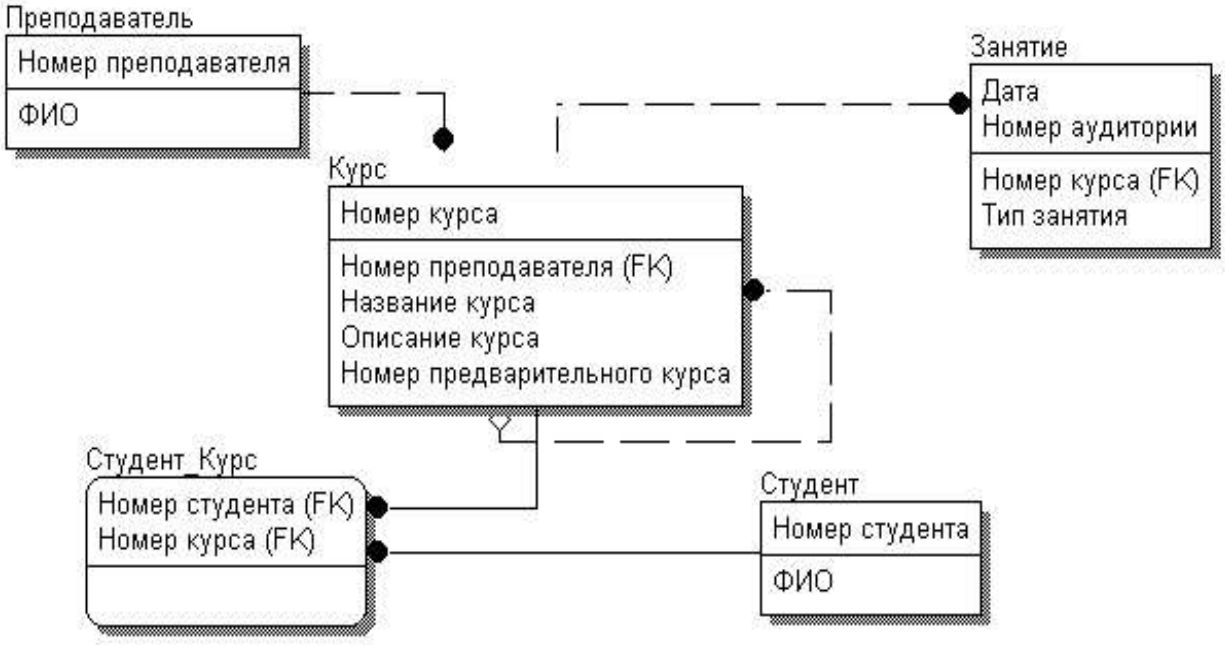


Рисунок А.2 – Структура системы, которая содержит информацию о внутренней системе обучения в большой промышленной компании

Вариант 3

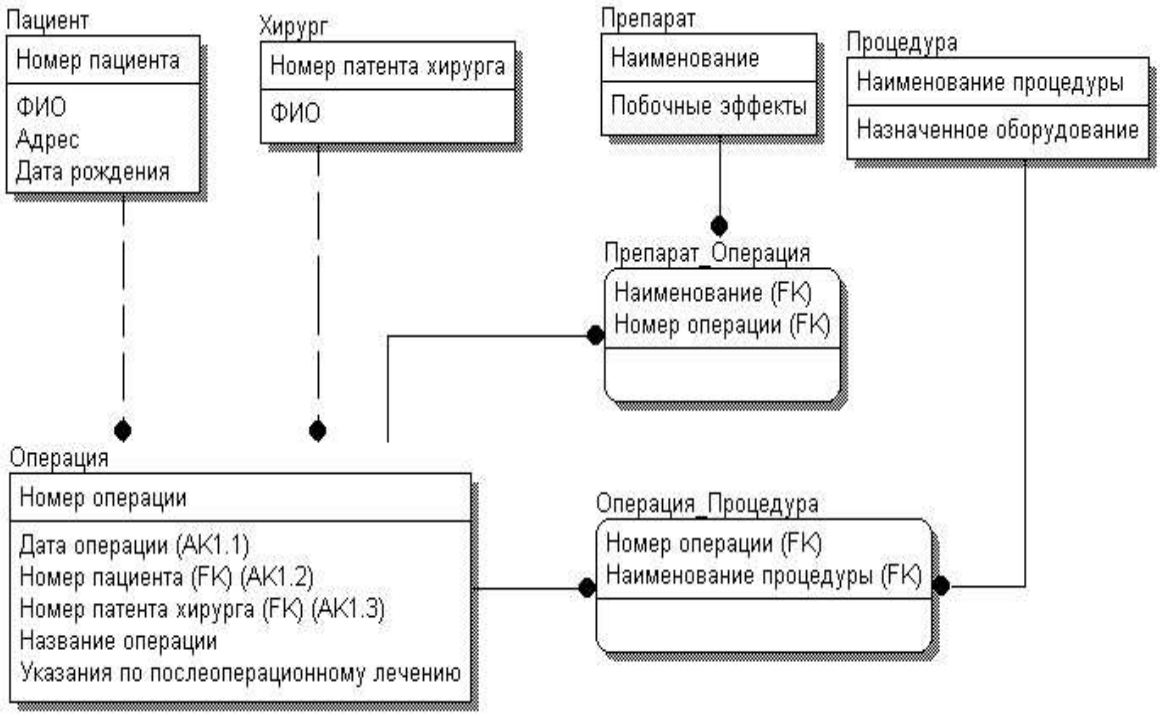
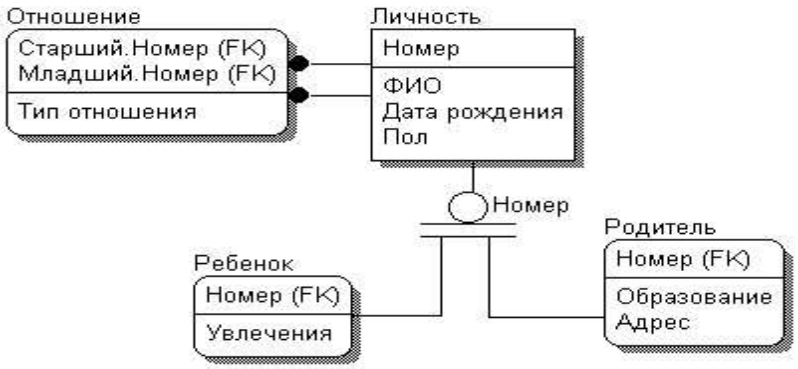


Рисунок А.3 – Структура системы, которая содержит информацию о пациентах клиники

Вариант 4



Типы отношений: состоят в браке, является родителем

Рисунок А.4 – Структура системы, которая содержит анкетные данные служащих конторы

Вариант 5

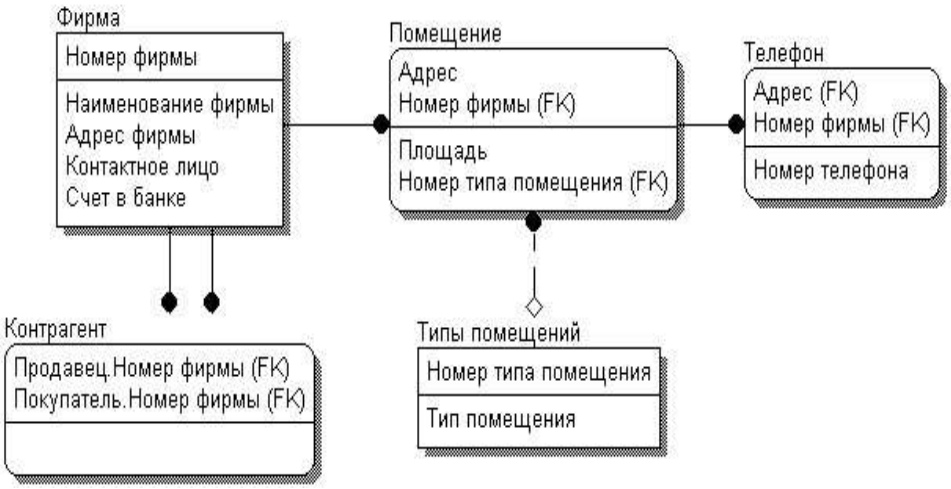


Рисунок А.5 – Структура системы, которая содержит информацию о торговых фирмах

Вариант 6

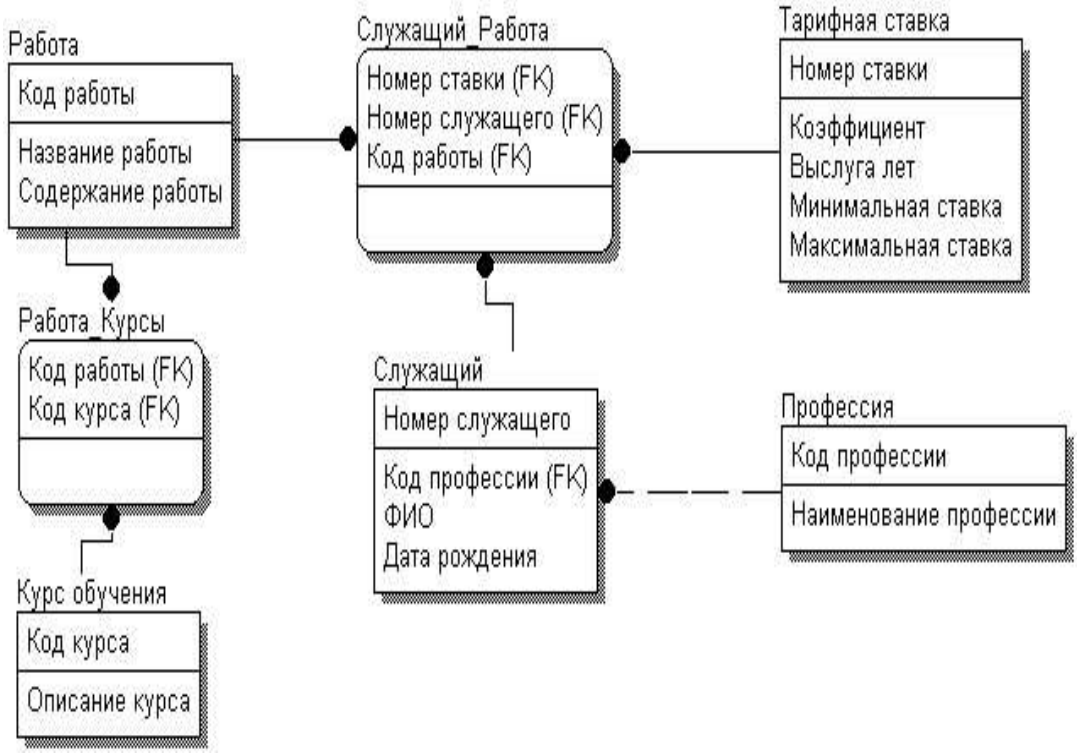
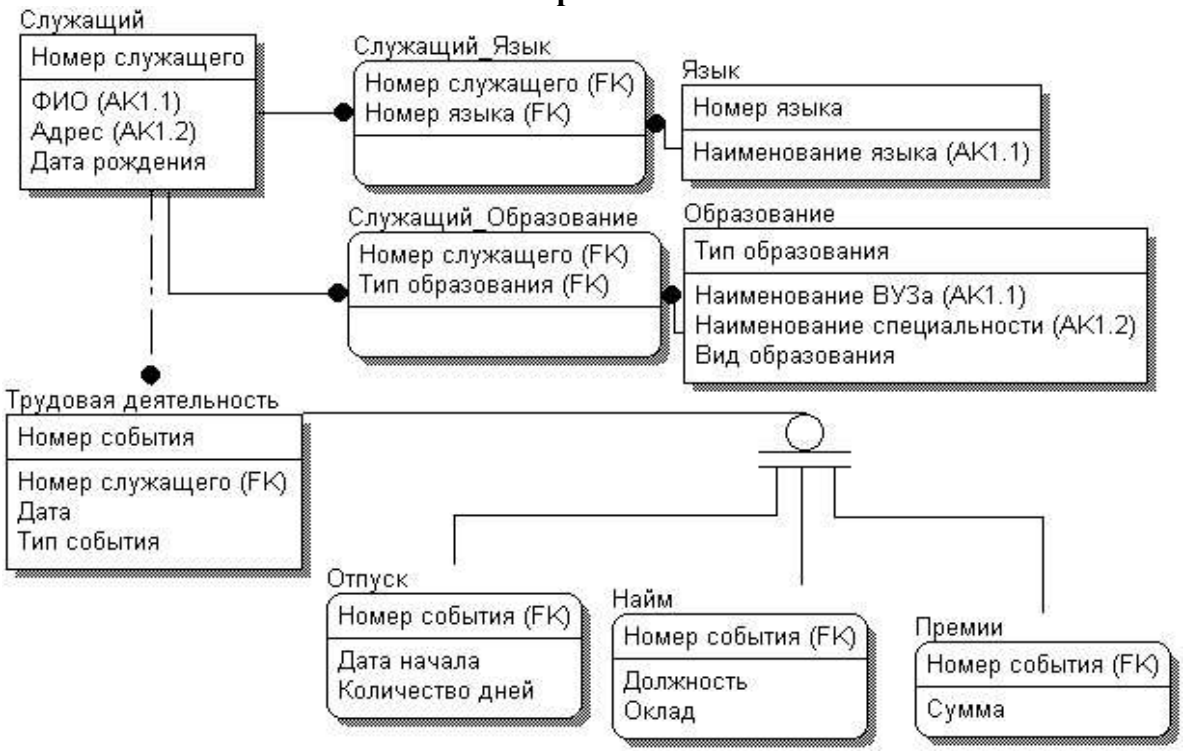


Рисунок А.6 – структура системы, которая содержит каталог работ

Вариант 7



Вид образования - высшее, среднее, неполное высшее.
Тип события – отпуск, премирование, события, связанные с наймом (в том числе изменение в должности и окладе).

Рисунок А.7 – Структура системы, которая содержит информацию о служащих

Вариант 8

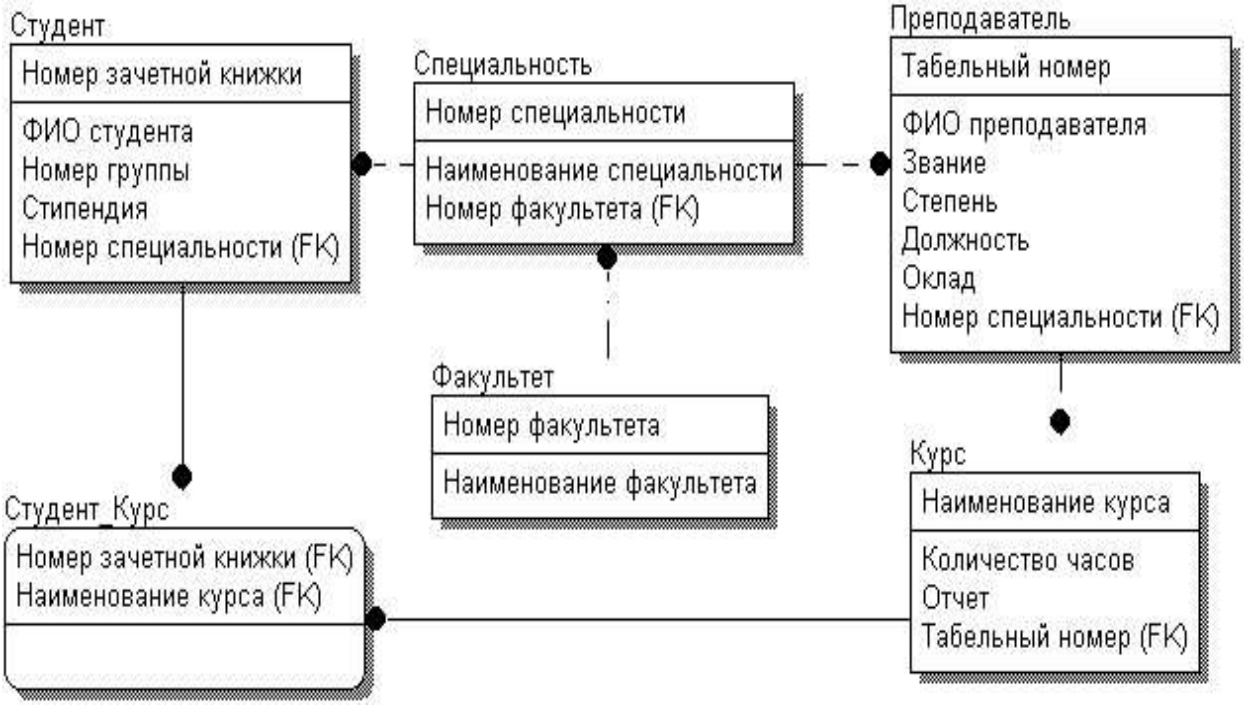


Рисунок А.8 – Структура системы, которая содержит информацию о факультетах

Вариант 9

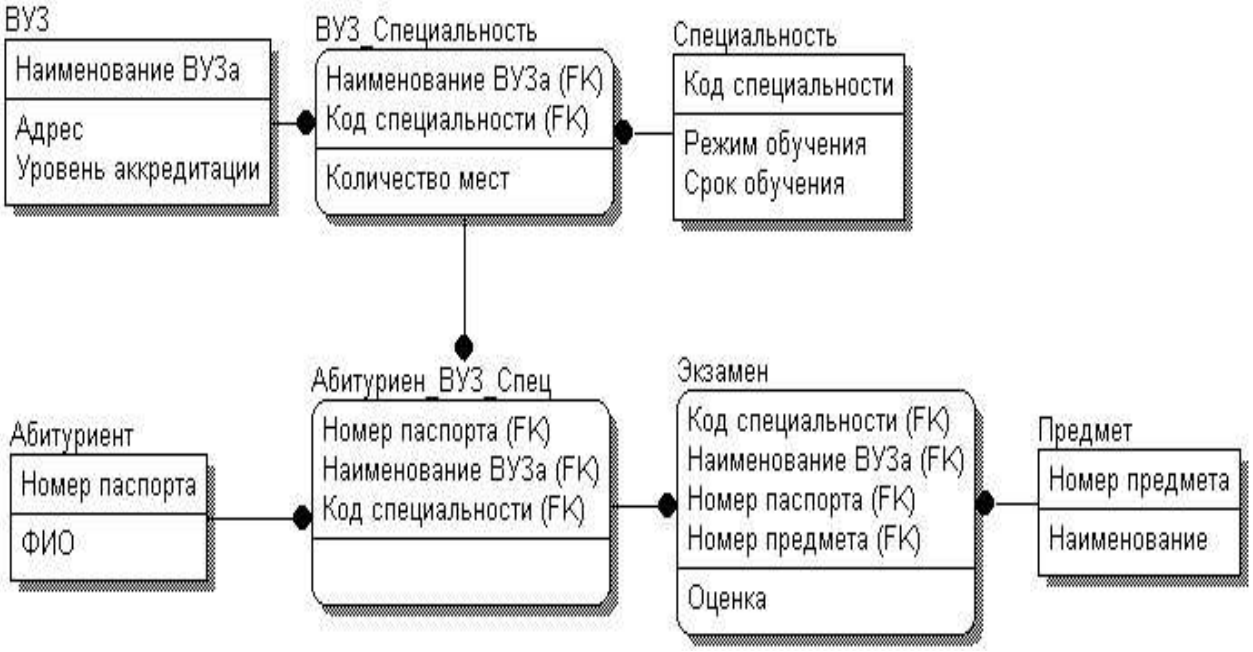
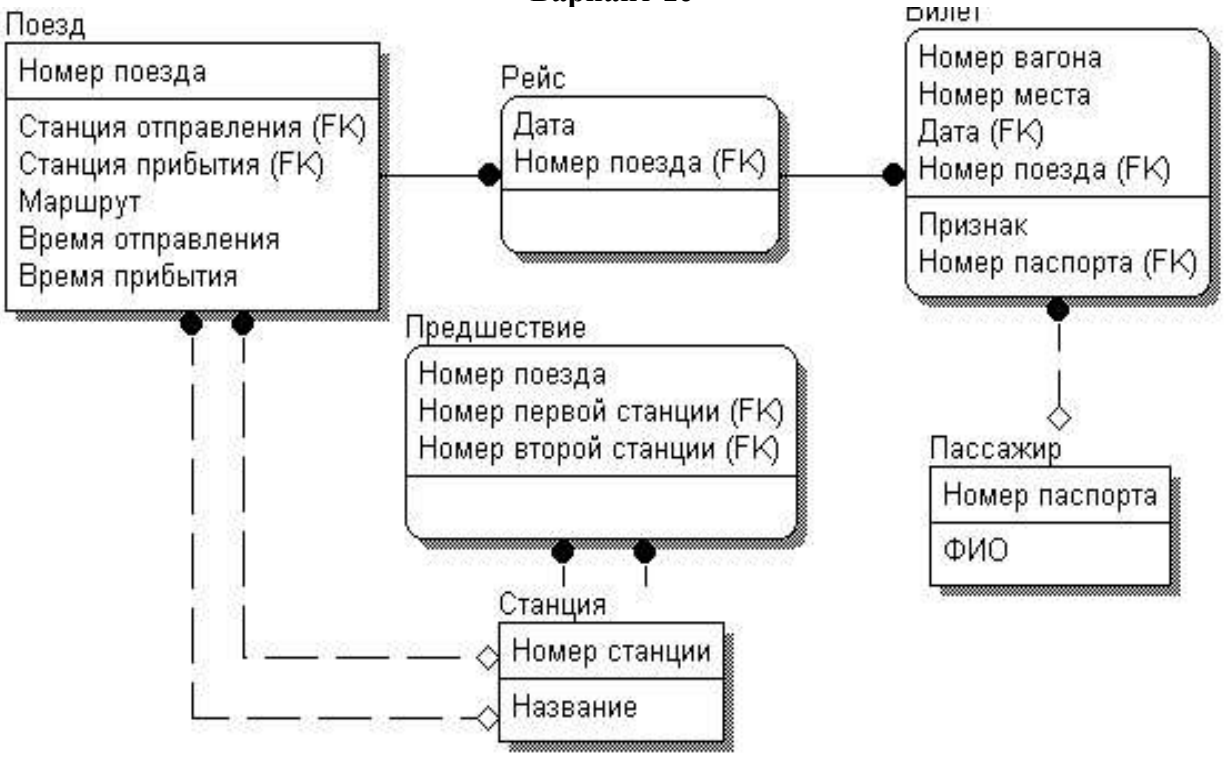


Рисунок А.9 – структура системы, которая содержит информацию об учебных заведениях, абитуриентах и экзаменах

Вариант 10



Признак билета – забронирован или выкуплен.

Рисунок А.10 – Структура системы, которая содержит информацию о движении поездов

Вариант 11

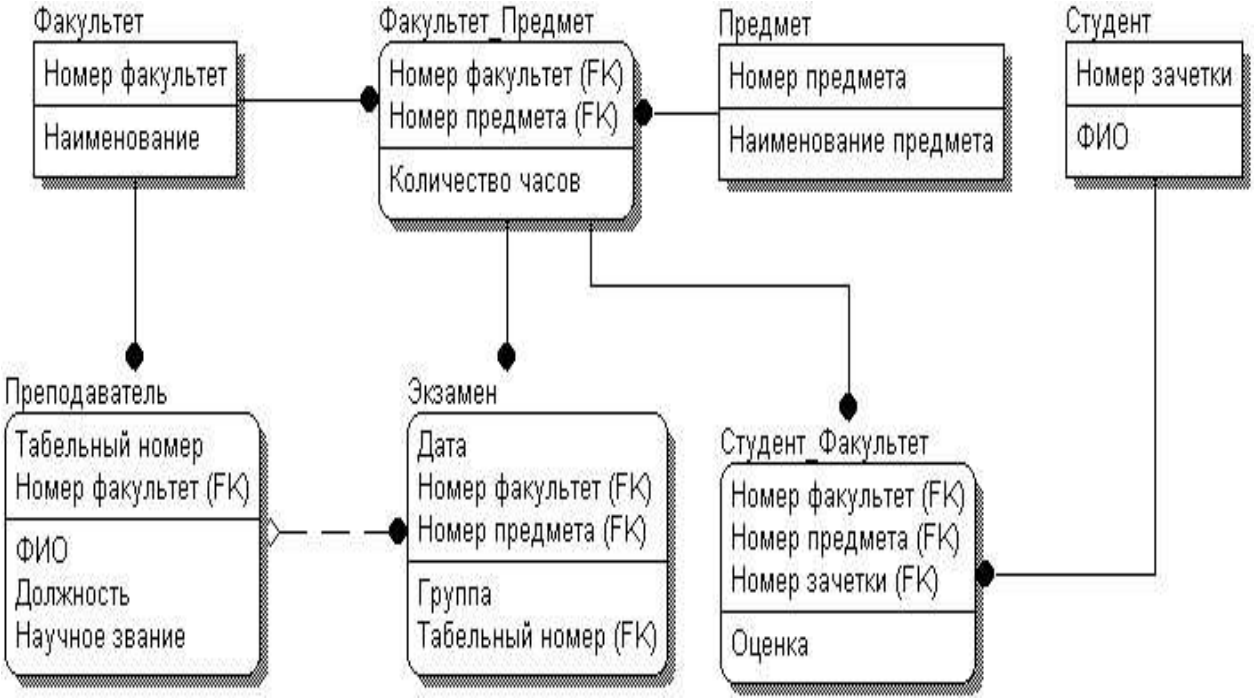


Рисунок А.11 – Структура системы, которая содержит информацию о результатах экзаменов по каждой кафедре

Вариант 12

Рисунок А.12 – Структура системы, которая содержит картотеку осужденных лиц

Вариант 13

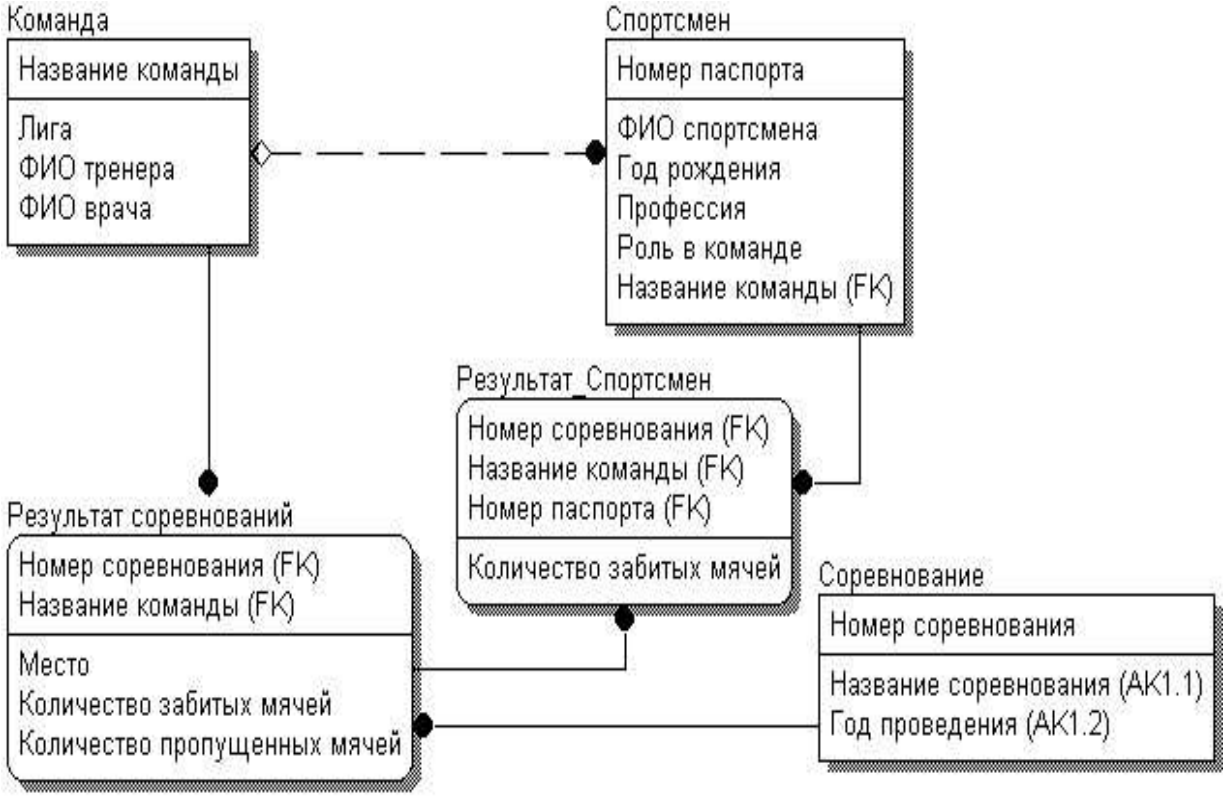
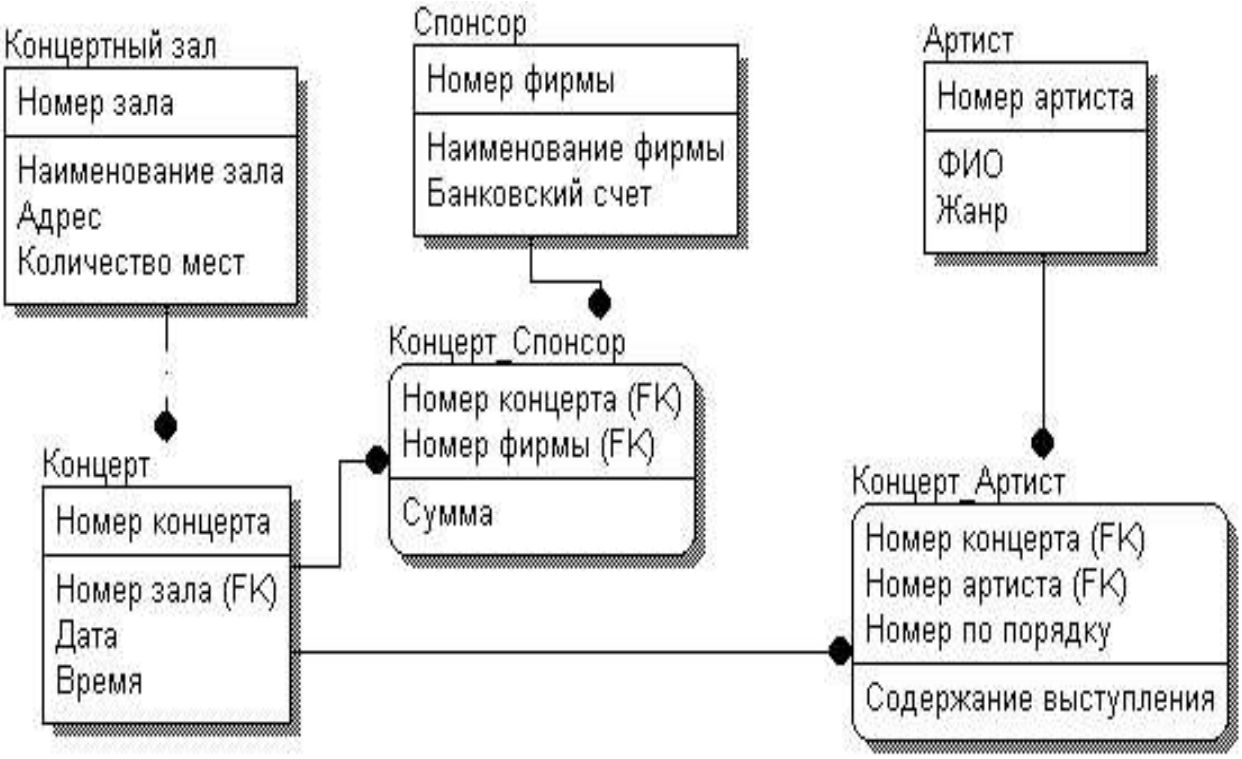


Рисунок А.13 – Структура системы, которая содержит информацию о футбольных командах

Вариант 14



Предусмотреть жанр «конферансье» – ведущий концерта.

Рисунок А.14 – структура системы, которая содержит информацию о концертах

Вариант 15

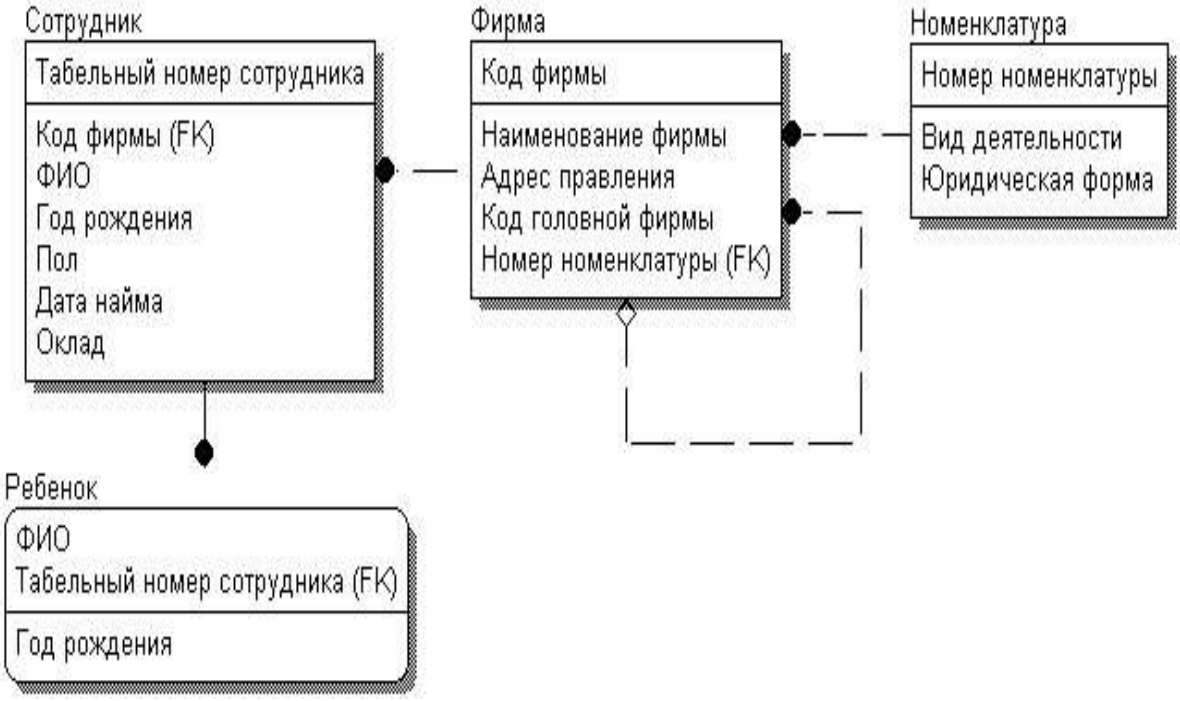
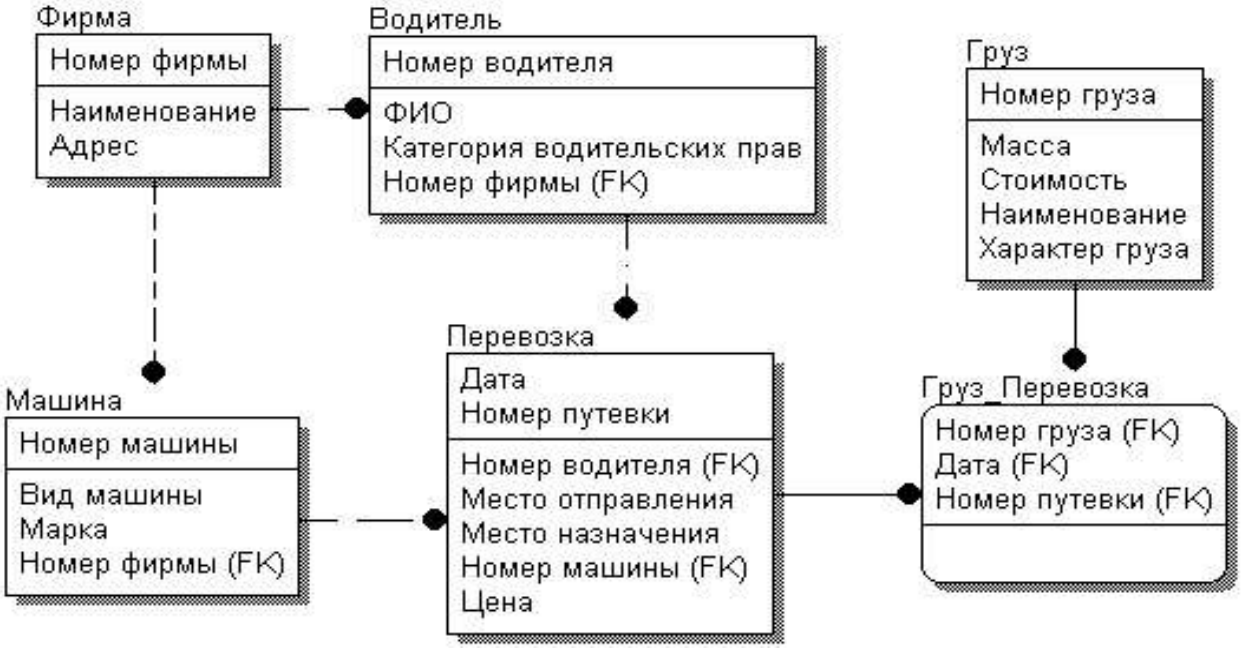


Рисунок А.15 – Структура системы, которая содержит картотеку фирм

Вариант 16



Категория водительских прав –A, B, C
Характер груза - твердый, жидкий и т.д.

Рисунок А.16 – Структура системы, которая содержит информацию о грузовых перевозках, осуществляемых различными фирмами

Вариант 17

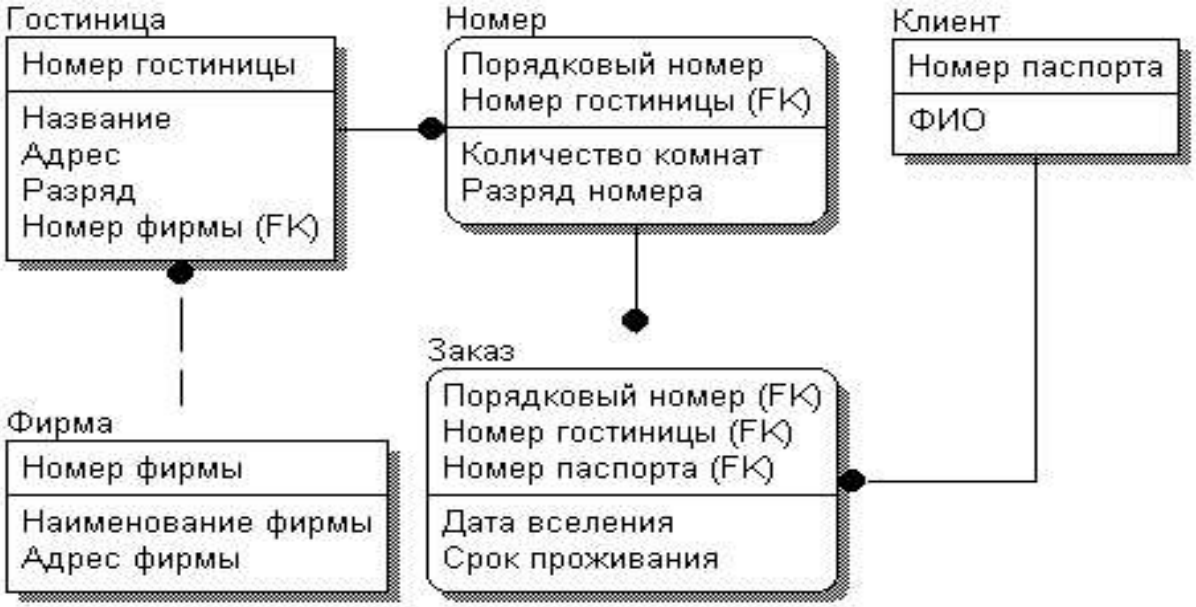


Рисунок А.17 – Структура системы, которая содержит информацию для бронирования мест в гостинице для командированных

Вариант 18

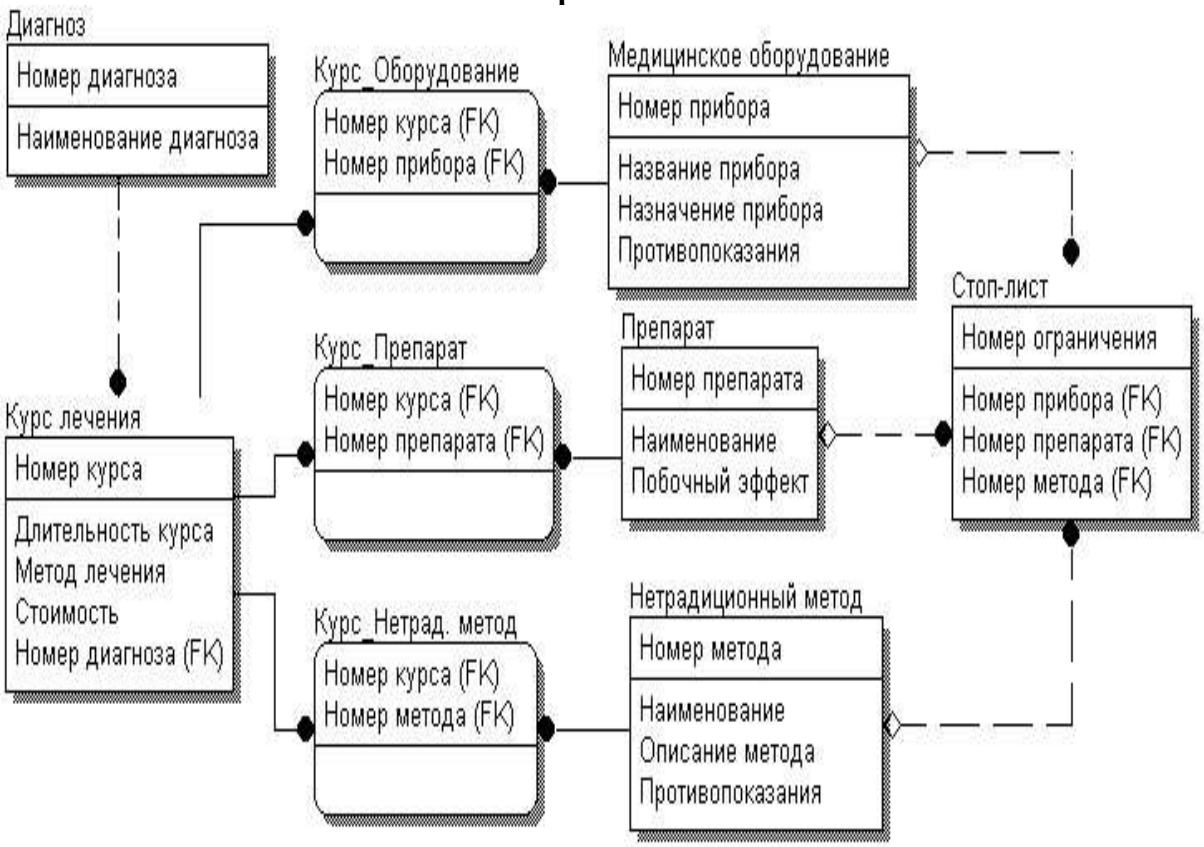


Рисунок А.18 – Структура системы, которая содержит информацию о лечении различных заболеваний

Вариант 19

Рисунок А.19 – Структура системы, которая содержит информацию о вычислительных центрах города

Вариант 20

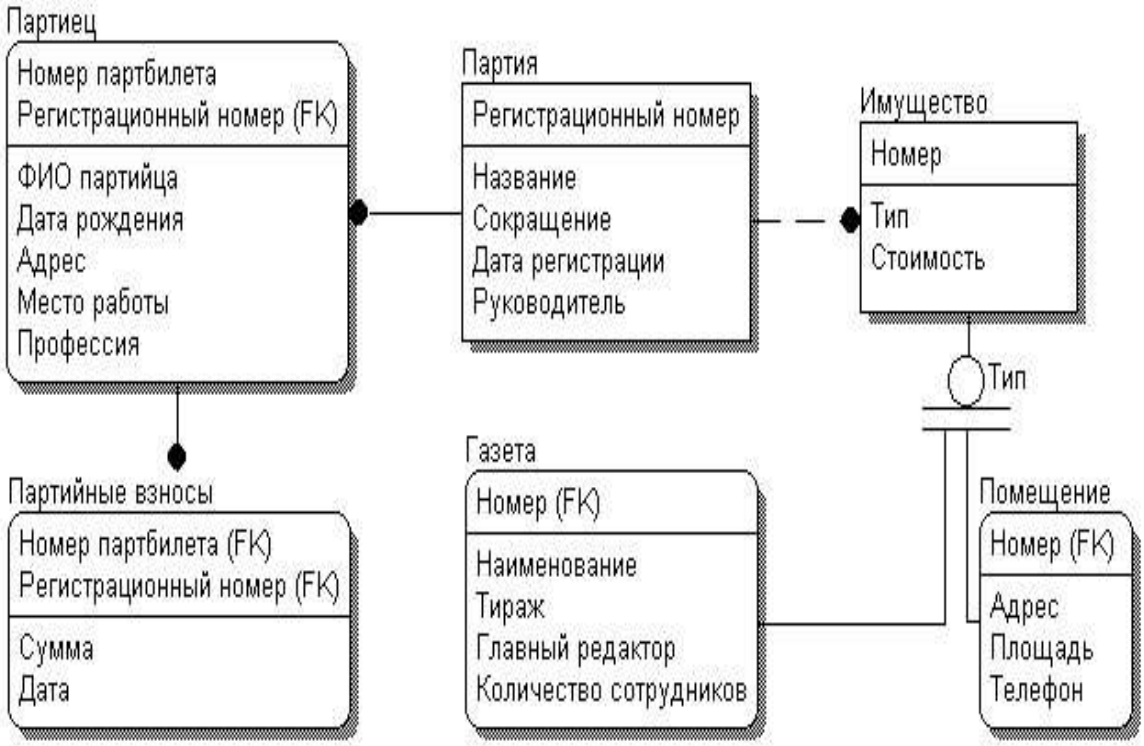


Рисунок А.20 – Структура системы, которая содержит информацию о политических партиях

ПРИЛОЖЕНИЕ Б
SQL

Синтаксис оператора CREATE DATABASE

```
CREATE {DATABASE | SCHEMA} '<спецификация файла>'
[USER '<имя пользователя>' [PASSWORD '<пароль>']]
[PAGE_SIZE [=] <целое>]
[LENGTH [=] <целое> [PAGE[S]]]
[DEFAULT CHARACTER SET <набор символов>]
[<вторичный файл>]...;

<вторичный файл> ::= FILE '<спецификация файла>'
[LENGTH [=] <целое> [PAGE[S]]]
[STARTING [AT [PAGE]] <целое>]
```

Таблица Б.1. – Описание конструкций оператора CREATE DATABASE

Аргумент	Описание
'<спецификация файла>'	Спецификация файла задает полный путь к создаваемому файлу базы данных и имя файла, включая его расширение. Сам файл должен отсутствовать на внешнем носителе.
USER '<имя пользователя>'	Имя пользователя-владельца базы данных. Может содержать до 31 символа. Нечувствительно к регистру.
PASSWORD '<пароль>'	Пароль пользователя-владельца базы данных. Может содержать до 64 символов, однако только первые 8 имеют значение. Чувствителен к регистру.
PAGE_SIZE	Задает размер страницы базы данных. Допустимы значения 4096 (значение по умолчанию), 8192 и 16384.Если вы зададите неправильное значение размера страницы, то система не выдаст сообщения об ошибке, а установит размер до ближайшего меньшего числа. Если указать значение меньше чем 4096, то будет выбрано значение по умолчанию — 4096.
LENGTH	Задает количество страниц в первичном файле базы данных.
DEFAULT CHARACTER SET	Задает набор символов по умолчанию для строковых типов данных в базе данных.
STARTING AT PAGE	Страница, с которой начинается вторичный файл базы данных.

Синтаксис оператора CONNECT

```
CONNECT '<спецификация файла>'
  [USER '<имя пользователя>' [PASSWORD '<пароль>']]
  [CACHE <целое> [BUFFERS]]
  [ROLE '<имя роли>'];
```

В операторе указывается имя первичного файла базы данных. Имя пользователя (предложение USER) и его пароль (PASSWORD) должны задавать пользователя, описанного в системе. Имя пользователя может содержать до 31 символа. Оно нечувствительно к регистру. Пароль чувствителен к регистру. Может содержать до 32 символов, однако только первые восемь имеют значение.

Предложение CACHE задает количество буферов кэш-памяти для соединения. По умолчанию 256. Перекрывает значение, указанное в конфиге или базе данных, только если больше указанных там значений.

Предложение ROLE задает имя роли, с которой пользователь соединяется с данной базой данных. Имя роли может содержать до 31 символа

Синтаксис оператора CREATE TABLE

```
CREATE TABLE <имя таблицы>
[EXTERNAL [FILE] '<спецификация файла>']
(<определение столбца>
[, <определение столбца> | <ограничение таблицы>]...);

<определение столбца> ::= <имя столбца>
{ <тип данных>
| <имя домена>
| COMPUTED [BY] (<выражение>)
| GENERATED ALWAYS AS (<выражение>)
}
[DEFAULT {<литерал> | NULL}]
[NOT NULL]
[<ограничение столбца>]
[COLLATE <порядок сортировки>]

<тип данных> ::=
{ SMALLINT [<размерность массива>]
| INTEGER [<размерность массива>]
| BIGINT [<размерность массива>]
| FLOAT [<размерность массива>]
| DOUBLE PRECISION [<размерность массива>]
| {DECIMAL | NUMERIC} [(<точность> [, <масштаб>])]
[<размерность массива>]
| {DATE | TIME | TIMESTAMP} [<размерность массива>]
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}
[(<целое>)] [CHARACTER SET <набор символов>]
[<размерность массива>]
| {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR}
[VARYING] [(<целое>)]}
[<размерность массива>]
| BLOB [SUB_TYPE {<номер подтипа> | <имя подтипа>}]
[SEGMENT SIZE <целое>]
[CHARACTER SET <набор символов>]}
| BLOB [(<размер сегмента> [, <номер подтипа>])]
}
<размерность массива> ::= [<целое 1>:<целое 2>
[, <целое 1>:<целое 2>]...]

<ограничение столбца> ::= [CONSTRAINT <имя ограничения>]
{ UNIQUE [<предложение USING>]
| PRIMARY KEY [<предложение USING>]
| REFERENCES <имя таблицы> [(<имя столбца>)]
[<предложение USING>]
[ON DELETE
{ NO ACTION
| CASCADE
| SET DEFAULT
| SET NULL
}
]
[ON UPDATE
{ NO ACTION
| CASCADE
| SET DEFAULT
| SET NULL
}
]
| CHECK (<условие столбца>)
}
<предложение USING> ::= USING
[ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

```
<условие столбца> ::=
{ <значение> <оператор сравнения>
{<значение> | (<выбор одного>)}
| <значение> [NOT] IN
(<значение> [, <значение>]... | <поиск одного>})
| <значение> [NOT] BETWEEN <значение> AND <значение>
| <значение> [NOT] LIKE <значение> [ESCAPE '<символ>']
| <значение> IS [NOT] NULL
| <значение> IS [NOT] DISTINCT FROM <значение>
| <значение> <оператор сравнения>
{ALL | SOME | ANY} (<поиск одного>)
| EXISTS (<поиск многих>)
| SINGULAR (<поиск многих>)
| <значение> [NOT] CONTAINING <значение>
| <значение> [NOT] STARTING [WITH] <значение>
| (<условие столбца>)
| NOT <условие столбца>
| <условие столбца> OR <условие столбца>
| <условие столбца> AND <условие столбца>
}
```

Описание конструкций оператора CREATE TABLE представлено в таблице Б.2.

Таблица Б.2 – Описание конструкций оператора CREATE TABLE

Аргумент	Описание
<имя таблицы>	Имя таблицы должно быть уникальным среди имен таблиц базы данных, хранимых процедур и представлений, описанных в этой базе данных
EXTERNAL [FILE] "<спецификация файла >"	Объявляет что данные для создаваемой таблицы, постоянно располагаются во внешней к базе данных таблице или файле.
<имя столбца>	Имя для столбца таблицы, должно быть уникальным именем в таблице.
<тип данных>	SQL тип данных столбца.
COMPUTED [BY] (<выражение>)	Указывает, что значение столбца вычисляется по другим столбцам. Выражение должно возвращать одиночное значение, и не иметь тип массива или возвращать массив.
<выражение>	Любое арифметическое выражение допустимое для типа данных столбца
COLLATE <порядок сортировки>	Определяет порядок сортировки для столбца. Порядок сортировки на уровне столбца отменяет порядок сортировки определенный на уроне домена.
DEFAULT	Определяет значение по умолчанию столбца, если оно не определяется каким-либо другим образом Значения: - literal: Вставляется указанная строка, числовое значение или дата. - NULL: Вводится значение NULL - USER: Вводится имя текущего пользователя. Столбец должен иметь тип, совместимый с текстовым Установка значению по умолчанию на уровне столбца отменяет значение по умолчанию на уровне домена
CONSTRAINT <имя ограничения>	Помещает именованное ограничение на таблицу или столбец. Если имя ограничения опущено, Firebird создает системное имя для ограничения.

Ограничение целостности CHECK

Синтаксис:

```
CHECK (<условие столбца>);
```

В условии можно пользоваться операторами сравнения, операторами NOT, BETWEEN, LIKE, IN, IS [NOT] NULL и прочими. У столбца может быть только одно условие CHECK.

Условие (<условие столбца>) должно принимать значение истинно, для того чтобы разрешить операцию добавления или изменения данных. <условие столбца> может проверять несколько значений, введенных в другие столбцы. Это ограничение на уровне таблицы.

Синтаксис оператора INSERT

```
INSERT INTO <object> [(col [, col ...])]
{VALUES (<val> [, <val> ...]) | <select_expr>}
[RETURNING <column_list> [INTO <variable_list>]];

<object> = tablename | viewname

val = {:variable | constant | expr
| function | udf ([val [, val …]])
| NULL | USER | RDB$DB_KEY | ?
} [COLLATE collation]

<constant> = num | 'string' | charsetname 'string'

<expr> = Допустимое выражение SQL, которое возвращает в одиночное значение столбца.

<function> = {
CAST (<val> AS <datatype>)
 | UPPER (<val>)
 | GEN_ID (generator, <val>)
 }

<select_expr> = SELECT возвращающий ноль или более строк,
 где число столбцов в каждой строке такое же,
 как число элементов, которые должны быть вставлены.
```

Описание конструкций оператора INSERT представлено в таблице Б.3.

Таблица Б.3 – Описание конструкций оператора INSERT

Аргумент	Описание
`INTO <object>`	Имя существующей таблицы или вида, в которую вставляются данные.
`col`	Имя существующего столбца в таблице или виде, в который вставляются значения.
`VALUES (<val> [, <val> …]`	Список значению для вставки в таблицу или вид. Значения должны быть в том же порядке, как целевые столбцы.
`RETURNING <column_list> [INTO <variable_list>]`	Только для Firebird 2.0 и выше.
`<select_expr>`	Запрос, который возвращает значения, для вставки в целевые столбцы.
`<column_list>`	Список возвращаемых доменов объекта, только Firebird v2.0 и выше
`<variable_list>`	Список переменных в которые будут помещены возвращаемые домены объекта. Только Firebird v2.0 и выше

Синтаксис оператора SELECT

```
<select statement> ::=
  <select expression> [FOR UPDATE] [WITH LOCK]
<select_expression> ::=
  <select_expr_body>
  [ ORDER BY <sort_value_list> ]
  [ ROWS     <record_number> [TO <record_number>] ]
```

```
<select_expr_body> ::=
  { <query_specification> | <select_expr_body> UNION [DISTINCT | ALL]
<query_specification> }

<query_specification> ::=
  SELECT
    [DISTINCT | ALL]
    [FIRST {<record_number> | (:param_recordnumber) } [SKIP <record_number>
| (:param_recordnumber) ] ]
    <select_list>
  FROM     <reference_expression_list>
  [ WHERE    <search condition>   ]
  [ GROUP BY <group_value_list>   ]
  [ HAVING   <group_condition>    ]
  [ PLAN     <plan_item_list>     ]

<select_list> ::=
  {  <constant>
   | <column_name>[<array_dim>]
   | <expression>
   | <function> ([<select_list>])
   | <UDF> ([<select_list>])
   | NULL
   | {CURRENT_USER | CURRENT_ROLE | CURRENT_TIMESTAMP | CURRENT_CONNECTION |
CURRENT_XXX }
   | RDB$DB_KEY
  } [COLLATE <collate>] [AS <column_alias_name>]

<reference_expression_list> ::=
  {<table name> | <view name> | <procedure name> | <joined table> | <derived
table> }

<derived table> ::=
  '(' <select expression> ')'

<joined table> ::=
  { <cross_join> | <qualified_join> }

<cross_join> ::=
  <reference_expression_list> CROSS JOIN <reference_expression_list>

<qualified_join> ::=
  <reference_expression_list> [{INNER | {LEFT | RIGHT | FULL} [OUTER]}] JOIN
<reference_expression_list> ON <join condition>

<search_condition> ::=
    <val> <operator> { <val> | <singletone_select> }
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING WITH <val>
  | <val> [NOT] IN ( { <val> [,<val> ] | <select expression> } )
  | <val> IS [NOT] NULL
  | {ALL | SOME | ANY} (<select expression>)
  | EXISTS (<select expression>)
  | SINGULAR (<select expression>)
  | {AND | OR} [NOT] (<search condition>)

<operator> ::=
  { = | < | > | <= | >= | !< | !> | <> | != }

<sort_value_list> ::=
  {<column_name> | <column_number | expression> } [COLLATE <collation> ]
[ASC[ENDING] | DESC[ENDING] [NULLS {FIRST|LAST}] ] [, <sort_value_list>]
```

Описание конструкций оператора SELECT представлено в таблице Б.4.

Таблица Б.4 – Описание конструкций оператора SELECT

Аргумент	Описание
SELECT [DISTINCT | ALL]	Определяет данные, для поиска. DISTINCT удаляет повторяющиеся значения из возвращенных данных. ALL, параметр по умолчанию, возвращает все данные.
*	возвращает все столбцы из определенной таблицы.
<select_list>	возвращает определенный список столбцов и значений
FROM <reference_expression_list>	Список таблиц, представлений и сохраненных процедур, из которых возвращаются данные. Список может включать JOIN, соединения могут быть вложенными.
<table name>	Имя существующей таблицы в базе данных.
<view name>	имя существующего представления в базе данных.
<procedure name>	Имя существующей сохраненной процедуры, которая функционирует, как инструкция SELECT.
<joined table>	Ссылка таблицы состоящая из JOIN.
WHERE <search_cond>	Определяет условия, которые ограничивают подмножество возвращаемых строк из всех доступных строк.
GROUP BY <group_value_list>	Разделяет результаты запроса в группы содержащие все строки с одинаковыми значениями, основанными на списке столбцов.
HAVING <group_condition >	Используется совместно с GROUP BY. Определяет условия, которые ограничивают группировку возвращаемых строк.
UNION	Комбинирует две или более таблиц, которые имеют полностью, либо частично одинаковую структуру.
ORDER BY <sort_value_list>	Определяет порядок в котором строки будут возвращены.

Оператор ALTER TABLE

Используется для модификации структуры существующей таблицы

Перед тем как использовать ALTER TABLE:

- сохраните данные таблицы;
- удалите все ограничения целостности со столбца.

Сохранение данных это процесс, состоящий из пяти шагов. Например, один из столбцов таблицы имеет тип CHAR(3), его надо поменять на CHAR(5).

Для этого надо:

1. Создать временный столбец, определив его так же, как существующий.
2. Переписать данные из существующего столбца во временный столбец
3. Модифицировать временный столбец
4. Переписать данные из временного столбца обратно
5. Уничтожить временный столбец

1. Добавляем новый столбец

```
ALTER TABLE EMPLOYEE ADD TEMP_NUM CHAR(3);
```

2. Переписываем

```
UPDATE EMPLOYEE SET TEMP_NUM = DEP_NUM;
```

3. Модифицируем

```
ALTER TABLE ALTER DEP_NUM TYPE CHAR(4);
```

4. Переписываем обратно

```
UPDATE EMPLOYEE SET DEP_NUM = TEMP_NUM;
```

5. Удаляем

```
ALTER TABLE DROP TEMP_NUM;
```

Удаление столбца

```
ALTER TABLE <имя таблицы> DROP <имя столбца>[, <имя столбца>...];
```

Например:

```
ALTER TABLE EMPLOYEE
DROP EMP_NUM,
DROP JOB_CODE;
```

Команда ALTER TABLE не сможет выполниться при следующих условиях:

- если данные в таблице после удаления нарушают ограничения PRIMARY KEY или UNIQUE;
- если столбец является частью первичного ключа, ограничения UNIQUE, или столбец является внешним ключом;
- столбец используется в ограничении CHECK;
- столбец используется в представлении (view), в триггере, или используется при вычислении другого поля.

В случае, если что-либо мешает удалить столбец, надо сначала удалить это «что-то», а затем удалять столбец.

Добавление столбца

```
ALTER TABLE <имя таблицы> ADD <определение столбца>
```

Например:

```
ALTER TABLE employee ADD emp_no INTEGER NOT NULL;
```

Добавление ограничения целостности на таблицу

```
ALTER TABLE <имя таблицы> ADD [CONSTRAINT <имя ограничения>]
<определение ограничения>;
```

Можно добавить ограничения PRIMARY KEY, FOREIGN KEY, UNIQUE, или CHECK.
Например:

```
ALTER TABLE EMPLOYEE ADD CONSTRAINT DEPT_NO UNIQUE(PHONE_EXT);
```

Удаление ограничения целостности, наложенного на столбец

```
ALTER TABLE <имя таблицы>
DROP CONSTRAINT <имя ограничения>;
```

Например:

```
ALTER TABLE PROJECT
DROP CONSTRAINT TEAM_CONSTRT;
```

Если имя ограничения небыло указано явно при создании таблицы, сервер генерирует имя автоматически. Посмотреть имена ограничений можно в системной таблице RDB$RELATION_CONSTRAINTS:

```
SELECT * FROM RDB$RELATION_CONSTRAINTS
```

Модификация столбца в таблице

Можно изменить имя столбца, тип столбца, позицию столбца в таблице.

```
ALTER TABLE <имя таблицы> ALTER [COLUMN] <имя столбца> <определение
изменеия>
```

<определение изменеия>:

- для имзменеия имени: TO <новое имя>
- для изменеия типа данных: TYPE <новый тип>
- для изменения позиции: POSITION <целое число>

Примеры:

1. Изменение позиции

```
ALTER TABLE EMPLOYEE ALTER EMP_NUM POSITION 2;
```

2. Изменение имени EMP_NUM на EMP_NO

```
ALTER TABLE EMPLOYEE ALTER EMP_NUM TO EMP_NO;
```

3. Изменение типа

```
ALTER TABLE EMPLOYEE ALTER EMP_NO TYPE CHAR(20);
```

Преобразование из не-символьных в символьные типы возможны при следующих ограничениях.

- BLOB и массивы не преобразуются.
- Новое определение поля должно вмещать старые данные (CHAR(3) не вместит CHAR(4)).

ПРИЛОЖЕНИЕ В
Ограничения целостности данных

Ограничения целостности - это правила, которые отслеживают достоверность данных. Ограничения целостности могут проверять значение, вводимое в один столбец (например, возраст не может быть отрицательным, и не может быть больше 150 лет), может проверять значение, вводимое в несколько столбцов, например, если в поле «возраст» введено значение 10 лет и менее, то в поле «количество детей» не может быть значение большее 0, может проверять несколько таблиц (например, при удалении заказчика надо удалить все его заказы). Ограничения, проверяющие значение, вводимое в столбец, называются ограничениями на уровне атрибута, проверяющие несколько столбцов – ограничения на уровне таблицы, проверяющие несколько таблиц – ограничения на уровне связи.

CREATE TABLE может создавать следующие типы ограничений целостности:

- PRIMARY KEY (первичный ключ) - уникально идентифицирует каждую строку таблицы. Значение в указанном столбце либо в упорядоченном наборе столбцов не могут повторяться в более чем одной строке. Столбец PRIMARY KEY должен быть определен только с атрибутом NOT NULL. Таблица может иметь только один PRIMARY KEY, который может быть определен на одном или более столбцов.
- UNIQUE (уникальные) ключи гарантируют, что не существует двух строк, имеющих одно и тоже значение в указанном столбце или упорядоченном наборе столбцов. Уникальный столбец должен быть определен с атрибутом NOT NULL. Таблица может иметь один или более UNIQUE ключей.

В случае, если первичный или альтернативный ключ состоит из одного поля, его можно описать в той же строке, что и само поле (такое ограничение называется ограничением на уровне столбца). Если первичный или альтернативный ключ состоит из нескольких атрибутов, он описывается после определения всех полей таблицы (такое ограничение называется ограничением на уровне таблицы).

Ограничение на уровне столбца имеет следующий синтаксис:

```
<col_constraint> = [CONSTRAINT <имя ограничения>] <определение ограничения>
```

т.е. определение ограничения состоит из необязательной части, состоящей из служебного слова CONSTRAINT и имени ограничения, и обязательной части, состоящей из собственно определения ограничения. В случае, если первая часть опущена, сервер генерирует имя ограничения автоматически. Посмотреть имя ограничения можно просмотрев системную таблицу RDB$RELATION_CONSTRAINTS.

Определение ограничения на уровне столбца описывается так:

```
<определение ограничения> = {
UNIQUE | PRIMARY KEY|CHECK (<условие проверки>)
| REFERENCES <ИмяДругойТаблицы> [( <имя столбца>[, <имя столбца> …])]
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
}
```

Поле можно определить как первичный ключ, его значение можно сделать уникальным по таблице, либо можно указать, что поле ссылается (REFERENCES) на некоторое поле из другой таблицы. Таблица, на которую ссылается поле, называется родительской. Смысл ограничения REFERENCES следующий: значение в поле в дочерней таблицы можно занести только в том случае, если это значение есть в соответствующем поле родительской таблицы. Например, если у нас есть таблица клиентов, и таблица покупок, поле “Номер клиента” из таблицы покупок должно ссылаться на поле “Номер клиента” из таблицы клиентов. Таким образом в таблицу покупое невозможно будет занести номер несуществующего клиента. Вторая часть описателя REFERENCES описывает, какие действия необходимо предпринять при

удалении из родительской таблицы (ON DELETE) и при обновлении родительской таблицы (ON UPDATE).

Всего возможно четыре варианта:

- NO ACTION - ничего не предпринимать;
- CASCADE – каскадировать. В случае удаления из родительской таблицы будут удалены все связанные записи из дочерней таблицы. Например, удаление клиента повлечет удаление всех его покупок. При модификации записи из родительской таблицы будут модифицированы также записи в дочерней таблице. Например, если клиент поменял номер с пятого на десятый, во всех его покупках поле “Номер клиента” также изменит свое значение с 5 на 10;
- SET DEFAULT – поле получает свое значение по умолчанию;
- SET NULL – поле устанавливается в NULL.

Какое именно действие необходимо предпринимать зависит от смысла таблиц. При удалении клиента удалять все его покупки имеет смысл. Но при удалении желтого цвета из таблицы цветов не стоит удалять все желтые автомобили.

Ограничения целостности на уровне таблицы описывается так же, как и ограничение на уровне столбца, за исключением определения самого ограничения .

```
<ограничение> = {{PRIMARY KEY | UNIQUE} ( <имя столбца> [,<имя столбца> …])
| FOREIGN KEY (<имя столбца> [,<имя столбца> …]) REFERENCES <ИмяДругойТаблицы>
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
| CHECK (<условие>)}
```

Отличие в том, что в PRIMARY KEY, UNIQUE и FOREIGN KEY можно описать более одного столбца.

Ссылочные ограничения гарантируют, что значения в наборе столбцов, которые определены в FOREIGN KEY принимают те же самые значения, которые присутствуют в столбце UNIQUE или PRIMARY KEY в таблице, на которую они ссылаются. Это ограничение на уровне связи.

1. Следующая инструкция создает одиночную таблицу с PRIMARY KEY:

```
CREATE TABLE COUNTRY(
 COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,
 CURRENCY VARCHAR(10) NOT NULL
);
```

2. Следующая инструкция создает таблицу с UNIQUE ограничением и на уровне столбца MODEL и на уровне таблицы (столбцы MODELNAME, ITEMID):

```
CREATE TABLE STOCK(
 MODEL SMALLINT NOT NULL UNIQUE,
 MODELNAME CHAR(10) NOT NULL,
 ITEMID INTEGER NOT NULL,
 CONSTRAINT MOD_UNIQUE UNIQUE (MODELNAME, ITEMID)
);
```

3. Следующая инструкция создает таблицу с вычисляемым столбцом NEW_SALARY:

```
CREATE TABLE SALARY_HISTORY(
EMP_NO EMPNO NOT NULL,
CHANGE_DATE DATE DEFAULT "NOW" NOT NULL,
UPDATER_ID VARCHAR(20) NOT NULL,
OLD_SALARY SALARY NOT NULL,
PERCENT_CHANGE DOUBLE PRECISION DEFAULT 0 NOT NULL
                CHECK(PERCENT_CHANGE BETWEEN -50 AND 50),
NEW_SALARY COMPUTED BY (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100),
PRIMARY KEY (EMP_NO, CHANGE_DATE, UPDATER_ID),
FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE (EMP_NO));
```

Пояснения к оператору. Объявлены поля:

поле **EMP_NO** целочисленное, не может принимать значение NULL.

поле **CHANGE_DATE** имеет тип DATE**,** значение по умолчанию – текущая дата, не может принимать значение NULL.

поле **UPDATER_ID** – строка переменной длины с максимальной длиной 20 символов, не может принимать значение NULL.

поле **OLD_SALARY** – содержит денежное значение, которое хранится с точностью до трех знаков после запятой, не может принимать значение NULL.

поле **PERCENT_CHANGE** вещественное число, по умолчанию имеет значение 0, не может принимать значение NULL.

Поле **NEW_SALARY** вычисляется как

```
OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100
```

Объявлены ограничения:

- ограничение CHECK – поле **PERCENT_CHANGE** может принимать значение из диапазона [-50,50]
- объявляется составной первичный ключ, состоящий из трех полей: **EMP_NO, CHANGE_DATE, UPDATER_ID.**
- описывается внешний ключ (**FOREIGN KEY**) - поле **EMP_NO** ссылается (**REFERENCES**) на поле **EMP_NO** таблицы **EMPLOYEE**, обновление и удаление из таблицы EMPLOYEE каскадируется.

ПРИЛОЖЕНИЕ Г
Варианты заданий к лабораторной работе №6

Таблица Г.1 – Задание на работу

Вариант	Задание
1	Создать представления «СписокСтатей», «СписокЖурналов» (объединяя таблицу «Публикация» с таблицами «Сведения о статье», «Сведения о журнале»), «Публикация» (Фамилия и инициалы автора, Наименование темы, Заглавие, Дата опубликования, Тип).
2	Создать представления Слушатели_курсов (Название курса, ФИО преподавателя, Количество студентов на курсе), Расписание (Название курса, Тип занятия, Дата, Номер аудитории)
3	Создать представления: Операция, подставив вместо номеров текстовые значения, Загрузка_хирурга(Фио хирурга, месяц, количество операций)
4	Представление объединяет таблицу «Личность» с таблицей «Ребенок» или «Родитель»
5	Создать представления: «ПомещениеТип» (в таблицу «Помещение» вместо номера типа помещения подставить наименование типа помещения), КлиентыФирмы(Наименование фирмы, Количество продавцов, Количество покупателей)
6	Создать представления: Рабочие (ФИО рабочего, Название работы, Выслуга лет, Минимальная ставка, Максимальная ставка)
7	Создать представления: «Список отпусков», объединив таблицы «Трудовая деятельность» и «Отпуск»; Премии(ФИО сотрудника, Сумма премий, Сумма дней отпусков)
8	Создать представления: Курсы (ФИО преподавателя, Наименование курса, Количество студентов), Факультеты(Наименование факультета, Количество курсов, Количество студентов, Количество преподавателей)
9	Создать представления: Экзамены (Код специальности, Наименование Вуза, ФИО абитуриента, Наименование предмета, Оценка), Успеваемость (Наименование Вуза, Код специальности, Процент пятерок, Процент четверок, Процент троек).
10	Создать представления: Расписание (Дата рейса, Номер поезда, Наименование станции отправления, Наименование станции прибытия, Количество проданных билетов); Пассажиры (Номер паспорта, Количество рейсов).
11	Создать представления: Экзамены (Дата, Наименование факультета, Наименование предмета, ФИО экзаменатора, Группа); Отметки (ФИО студента, Название предмета, Оценка).
12	Создать представления: Псевдонимы (Номер персоны, ФИО, Псевдоним), Дела (Номер дела, Описание, Осуждено персон по делу, Суммарный срок, Средний срок).
13	Создать представления: Результаты (Название соревнований, ФИО спортсмена, Количество забитых мячей), Лучшие (Название команды, Название соревнования, ФИО спортсмена, забившего больше всего голов).
14	Создать представления: Программа_концерта (Дата концерта, Время концерта, Номер зала, ФИО артиста, Номер по порядку в концерте, Содержание выступления); Артисты (ФИО артиста, количество выступлений в год).
15	Создать представления: Родственники (Табельный номер сотрудника, ФИО сотрудника, ФИО ребенка, Год рождения ребенка); Дочерние_фирмы (Название головной фирмы, Название дочерней фирмы).
16	Создать представления: Перевозки (Дата, Номер путевки, ФИО водителя, Место отправления, Место назначения, Количество грузов);
17	Создать представления: Номера_гостиницы (Название гостиницы, Адрес, Разряд, Порядковый номер номера, Количество комнат, Разряд номера); Клиенты (ФИО клиента, Количество заказов, Суммарный срок проживания).
18	Создать представления: Сложность (Наименование диагноза, Количество единиц оборудования, Количество препаратов, Количество нетрадиционных методов); КурсПрепарат (Номер курса, Побочный эффект).
19	Создать представления: Машины_инженера, состоящее из всех не ключевых полей таблиц «Инженер» и «Машина»; Комплектующие (Номер машины, Стоимость комплектующих)
20	Создать представления: Партии (Название партии, количество партийцев, Суммарная стоимость имущества); Газеты (Наименование, Тираж, Главный редактор, Количество сотрудников, Стоимость).

ПРИЛОЖЕНИЕ Д
Варианты заданий к лабораторной работе №8

Таблица Д.1 – Задание на работу

Вариант	Задание
1	Создать генераторы для полей «Номер автора», «Номер темы», «Идентификатор». Автоматическая генерация идентификатора публикации. Создать обновляемые VIEW «СписокСтатей» и «СписокЖурналов».
2	Создать генераторы для полей «Номер преподавателя», «Номер курса», «Номер студента». Автоматическая генерация полей «Номер преподавателя», «Номер курса», «Номер студента». Не позволять студенту посещать более 5 курсов (выдавать сообщение).
3	Создать генераторы для полей «Номер пациента», «Номер патента хирурга», «Номер операции». Автоматическая генерация полей «Номер пациента», «Номер патента хирурга», «Номер операции». Не позволять назначать более пяти процедур на операцию.
4	Создать генератор для поля «Номер». Создать представления: «СписокДетей», «СписокРодителей» (представление объединяет таблицу «Личность» с таблицей «Ребенок» или «Родитель»). Автоматическая генерация поля «Номер». Создать обновляемые VIEW «СписокДетей», «СписокРодителей», используя триггеры.
5	Создать генераторы для полей «Номер фирмы», «Номер типа помещения». Автоматическая генерация полей «Номер фирмы», «Номер типа помещения», создать обновляемое представление «ПомещениеТип» (в таблицу «Помещение» вместо номера типа помещения подставить наименование типа помещения).
6	Создать генераторы для полей «Код курса», «Номер служащего», «Код профессии», «Номер ставки», «Код работы». Автоматическая генерация полей «Код курса», «Номер служащего», «Код профессии», «Номер ставки», «Код работы». Не позволять служащему работать более чем на двух работах.
7	Создать генераторы для полей «Номер служащего», «Номер события», «Номер языка», «Тип образования». Автоматическая генерация полей «Номер служащего», «Номер события», «Номер языка», «Тип образования». Создать редактируемое представление «Список отпусков», объединив таблицы «Трудовая деятельность» и «Отпуск»
8	Создать генераторы для полей «Номер специальности», «Номер факультета». Автоматическая генерация полей «Номер специальности», «Номер факультета». Не позволять преподавателю читать более пяти курсов.
9	Создать генераторы для полей «Номер предмета», «Код специальности». Автоматическая генерация поля «Номер предмета», создать редактируемое представление «Экзамены», состоящие из полей «Код специальности», «Наименование ВУЗа», «ФИО абитуриента», «Наименование предмета», «Оценка».
10	Создать генераторы для полей «Номер поезда», «Номер станции». Автоматическая генерация полей «Номер поезда», «Номер станции». Создать редактируемое представление «Расписание», состоящие из полей: «Дата рейса», «Номер поезда», «Наименование станции отправления», «Наименование станции прибытия», «Количество проданных билетов».
11	Создать генераторы для полей «Номер факультета», «Номер предмета». Автоматическая генерация полей «Номер факультета», «Номер предмета». Не разрешать студенту сдавать экзамены на двух факультетах.
12	Создать генераторы для полей «Номер персоны», «Номер дела», «Номер псевдонима». Автоматическая генерация полей «Номер персоны», «Номер дела», «Номер псевдонима». Создать обновляемое представление «Псевдонимы», состоящее из полей «Номер персоны», «ФИО», «Псевдоним».
13	Создать генераторы для полей «Номер соревнования», «Количество забитых мячей». Автоматическая генерация поля «Номер соревнования». Не допускать проведения в год более пяти соревнований.
14	Создать генераторы для полей «Номер зала», «Номер концерта», «Номер фирмы», «Номер артиста». Автоматическая генерация полей «Номер зала», «Номер концерта», «Номер фирмы», «Номер артиста». Создать обновляемое представление «Программа концерта», состоящее из полей «Дата концерта», «Время концерта», «Номер зала», «ФИО артиста», «Номер по порядку в концерте», «Содержание выступления».
15	Создать генераторы для полей «Код фирмы», «Номер номенклатуры». Автоматическая генерация полей «Код фирмы», «Номер номенклатуры». Создать обновляемое представление «Родственники», состоящее из полей «Табельный номер сотрудника», «ФИО сотрудника», «ФИО ребенка», «Год рождения ребенка».
16	Создать генераторы для полей «Номер фирмы», «Номер водителя», «Номер груза». Автоматическая генерация полей «Номер фирмы», «Номер водителя», «Номер груза». Не позволять водителю делать более пяти перевозок в день.
17	Создать генераторы для полей «Номер гостиницы», «Номер фирмы». Автоматическая генерация полей «Номер гостиницы», «Номер фирмы». Создать обновляемое представление «Номера гостиницы», состоящее из полей «Название гостиницы», «Адрес», «Разряд», «Порядковый номер номера», «Количество комнат», «Разряд номера».

Продолжение таблицы Д.1

Вариант	Задание
18	Создать генераторы для полей «Номер диагноза», «Номер курса», «Номер прибора», «Номер препарата», «Номер метода», «Номер ограничения». Автоматическая генерация полей «Номер диагноза», «Номер курса», «Номер прибора», «Номер препарата», «Номер метода», «Номер ограничения». Не позволять на курс лечения назначать более пяти препаратов.
19	Создать генераторы для полей «Номер инженера», «Номер ВЦ», «Номер пакета», «Номер комплектующего». Автоматическая генерация полей «Номер инженера», «Номер ВЦ», «Номер пакета», «Номер комплектующего». Создать редактируемое представление «Машины инженера», состоящее из всех не ключевых полей таблиц «Инженер» и «Машина».
20	Создать генераторы для полей «Номер имущества», «Номер помещения», «Номер газеты». Автоматическая генерация полей «Номер имущества», «Номер помещения», «Номер газеты». Не разрешать партии владеть более чем пятью газетами.

УПРАВЛЕНИЕ ДАННЫМИ

Учебно-методическое пособие

Составители: **Доронина** Юлия Валентиновна,
Волкова Анастасия Викторовна

В авторской редакции

Изд. № 134/19. Объем 9 п.л.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»