

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)**

Факультет Информатика и вычислительная техника

(наименование факультета)

Кафедра Кибербезопасность информационных систем

(наименование кафедры)

**ОТЧЕТ**

по лабораторным работам по дисциплине “Система управления базами данных”

Выполнил:

ст. гр. ВКБ31 Котелевец К.А.

Проверил:

доц. Савельев В.А.

Ростов-на-Дону

2024

**ЛАБОРАТОРНАЯ РАБОТА №1**

**Основные понятия реляционных баз данных.**

**Создание таблиц в Microsoft Access.**

**Цель**: Знакомство с основными понятиями управления базами данных, структуре и приемами формирования их при работе в Microsoft Access. Создание учебной базы данных. Знакомство с основными приемами работы конструктора базы данных. Ввод данных в табличном режиме.

**Ход работы:**

**Задание 1.**

У меня MS Access 2010, работать буду с расширением базы данных .mdb

Запустим Microsoft Access и увидим интерфейс программы на рисунке 1.1

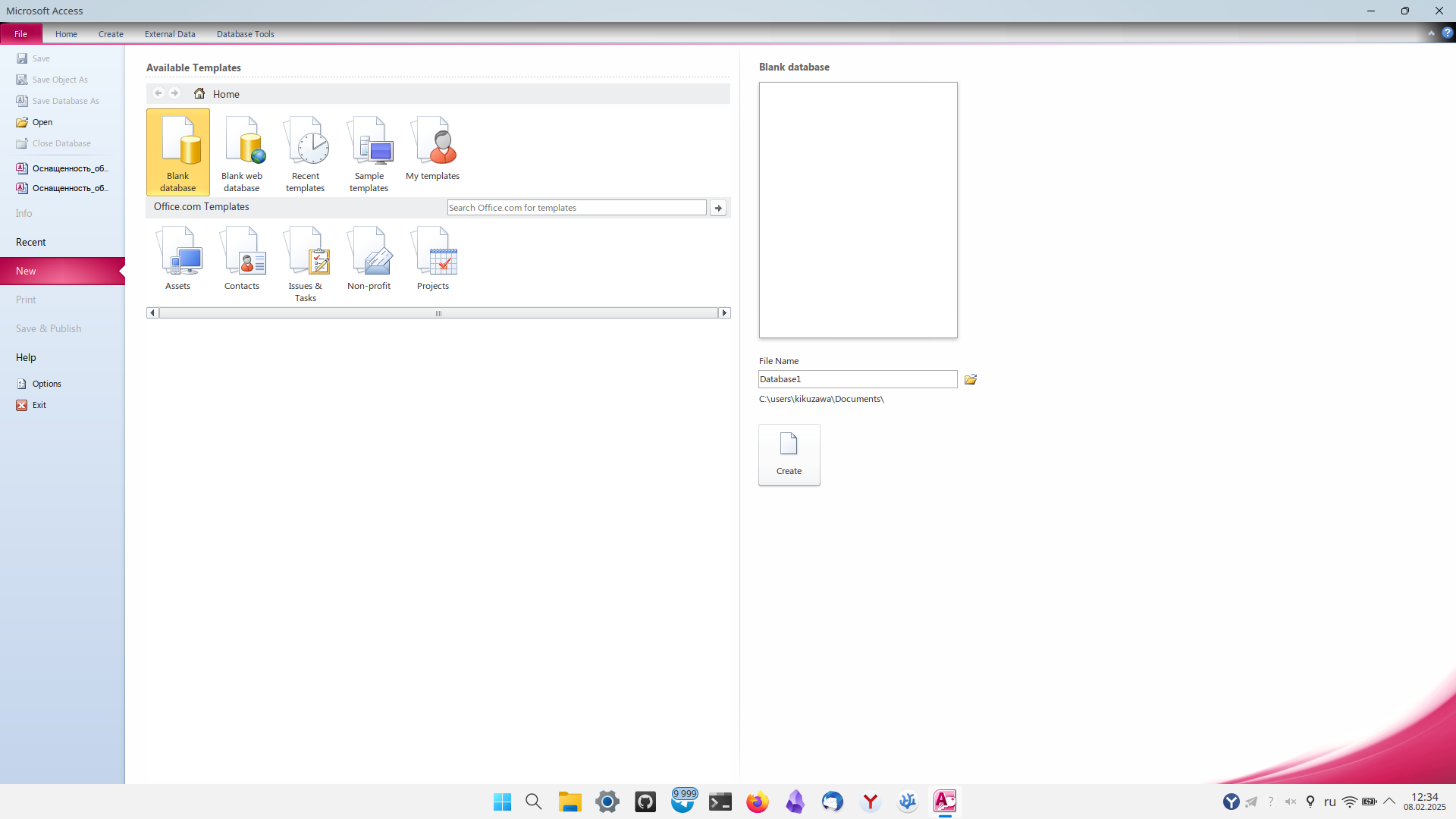


Рисунок 1.1 – Интерфейс программы MS Access

Создаем новую базу данных и назовем ее УчБД (Я ее назову UcDB). Можно увидеть это на рисунке 1.2

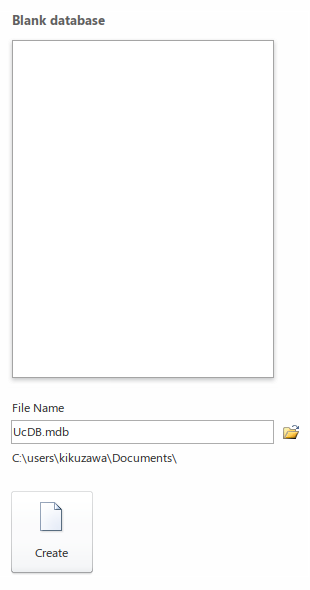


Рисунок 1.2 – Создание базы данных

Теперь запустим нашу БД и увидем интерфейс работы с ней на рисунке 1.3

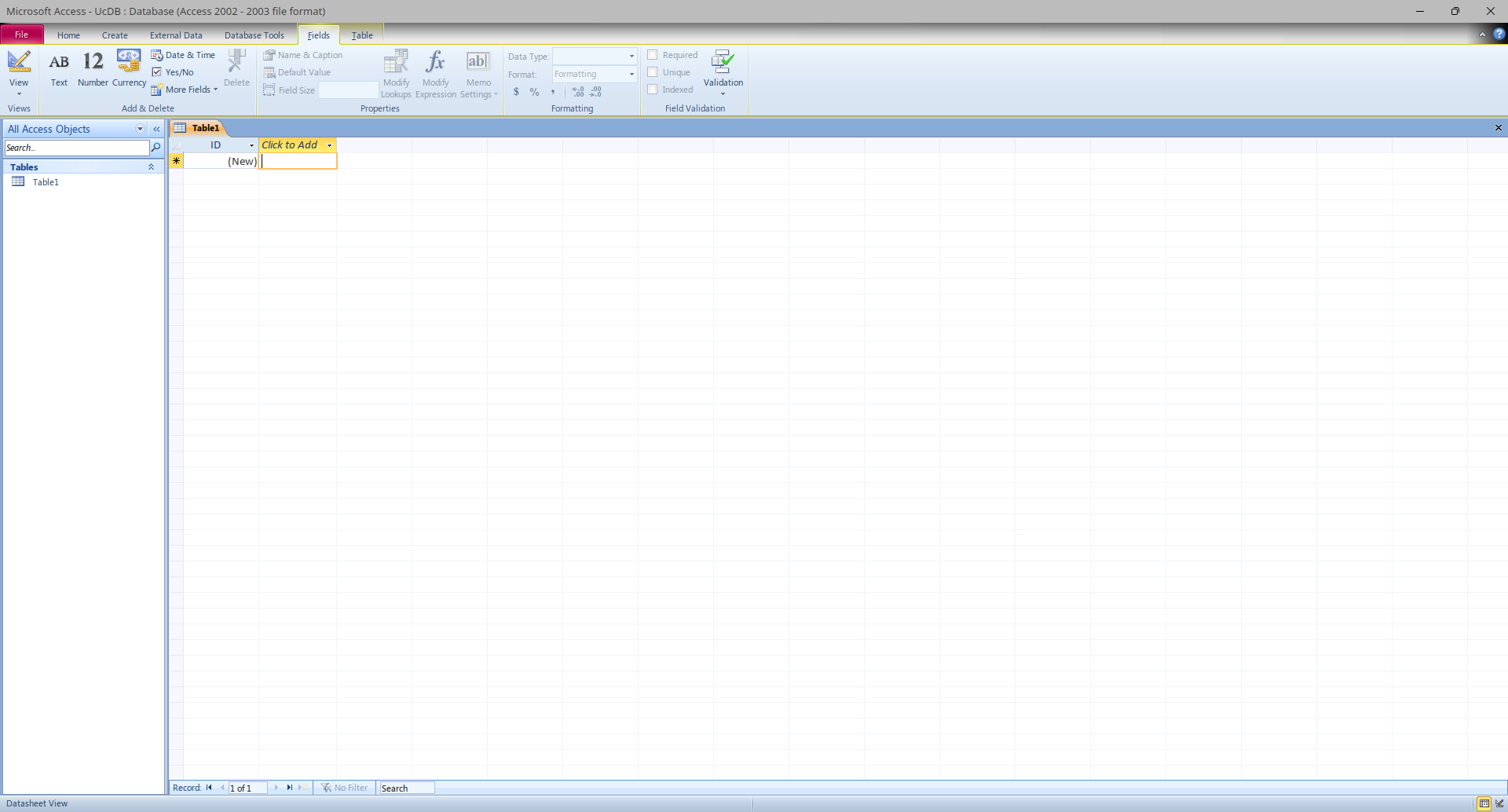


Рисунок 1.3 – Интерфейс работы с БД

Начнём с создания таблиц:

1. Список;
2. Паспорт;
3. Образование;
4. Дети;
5. Оплата.

На рисунках 1.4-1.8 можно увидеть уже созданные мной таблицы в формате конструктора.

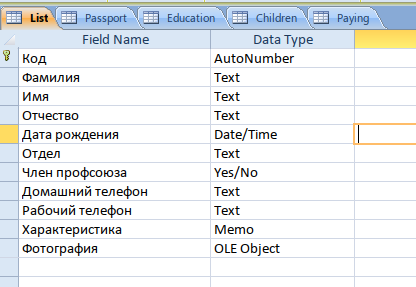


Рисунок 1.4 – Таблица в режиме конструктора Список

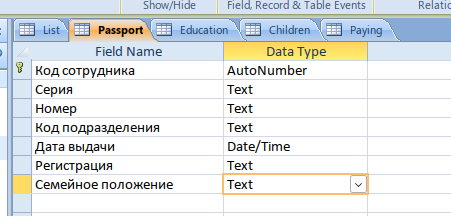


Рисунок 1.5 – Таблица в режиме конструктора Паспорт

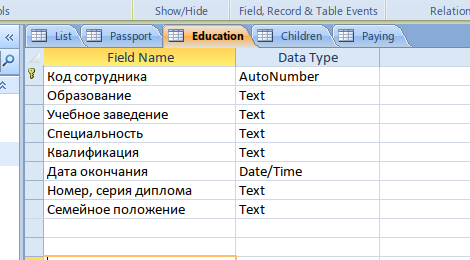


Рисунок 1.6 – Таблица в режиме конструктора Образование

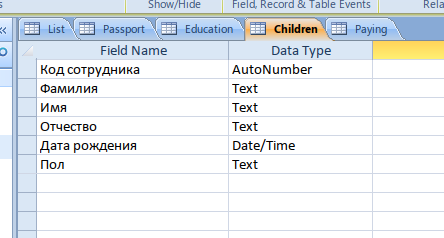


Рисунок 1.7 – Таблица в режиме конструктора Дети

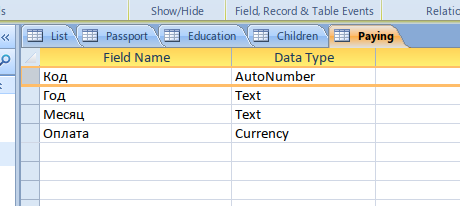


Рисунок 1.8 – Таблица в режиме конструктора Оплата

**Задание 2. Создание связей**

Необходимо установить связи между таблицами. Для этого нужно работать в окне “Схема данных”. В английской версии она называется “Relationships”. На рисунке 1.9 можно увидеть ее интерфейс

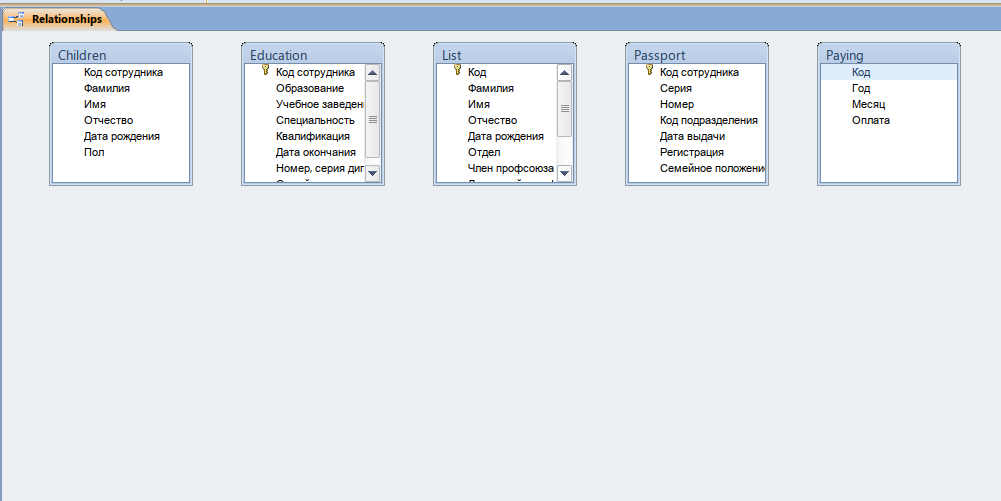


Рисунок 1.9 – Схема данных

На рисунке 1.10 происходит процесс создания связи в примере с детьми

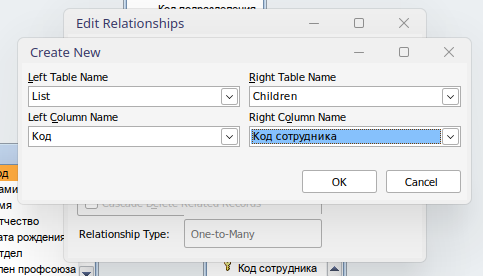


Рисунок 1.10 – Создание связи

Проделаем это всё с другими таблицами и получаем схему, представленную на рисунке 1.11

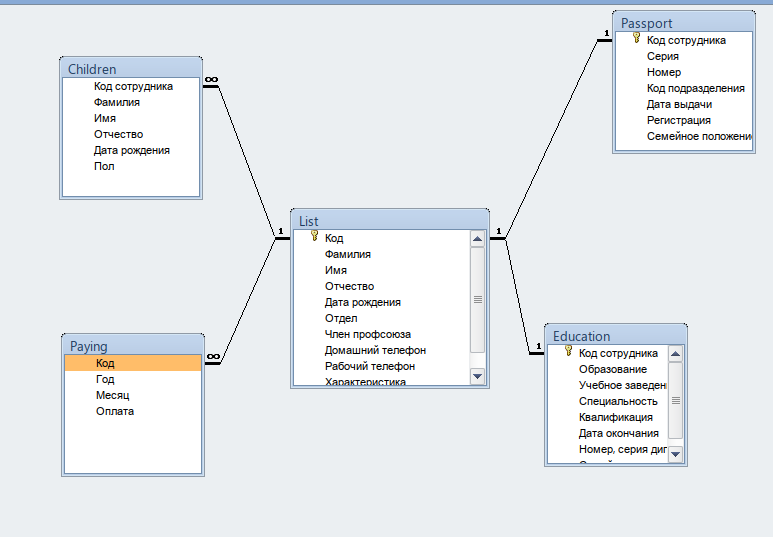


Рисунок 1.11 – Схема связи

**Контрольные вопросы:**

1. Типы отношений между таблицами?

* **Один к одному (1:1)**
* **Один ко многим (1:N)**
* **Многие ко многим (N:M)**

1. Что необходимо для обеспечения целостности данных?

* Primary Key
* Каскадное обновление и удаление
* Обеспечение целостности данных при создании связей
* **Ограничения на данные (Правила валидации)**

1. Поясните параметры «Каскадное удаление связанных записей»,

«Каскадное обновление связанных полей»

* **Каскадное удаление**  
  При удалении записи в родительской таблице **автоматически удаляются** все связанные записи в дочерней таблице.
* **Каскадное обновление**  
  При изменении значения первичного ключа в родительской таблице **автоматически обновляются** все связанные внешние ключи в дочерней таблице.

**Задание 3. Запросы**

1. Создайте запрос по таблице Список и найдите записи код которых равен 1.2.3. 8.9. (на рисунке 1.12-1.13)

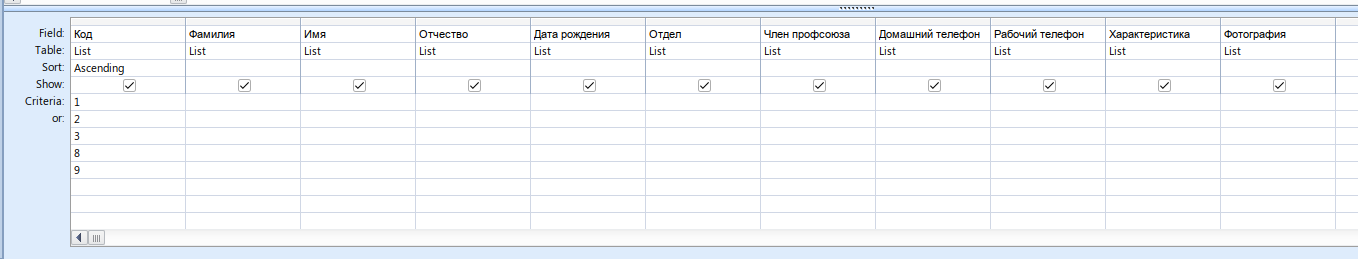


Рисунок 1.12

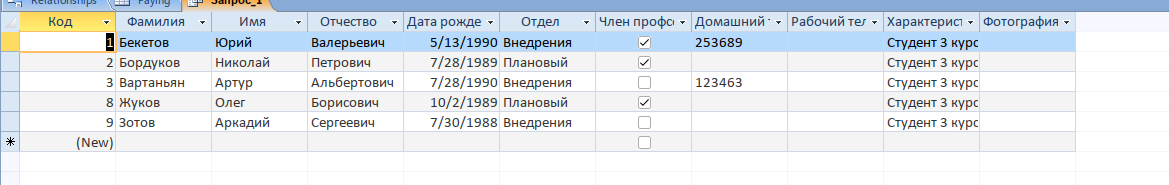


Рисунок 1.13

1. Создайте запрос по таблице Список и найдите записи относящиеся к одному отделу, а затем к другому. (на рисунках 1.14-1.17)

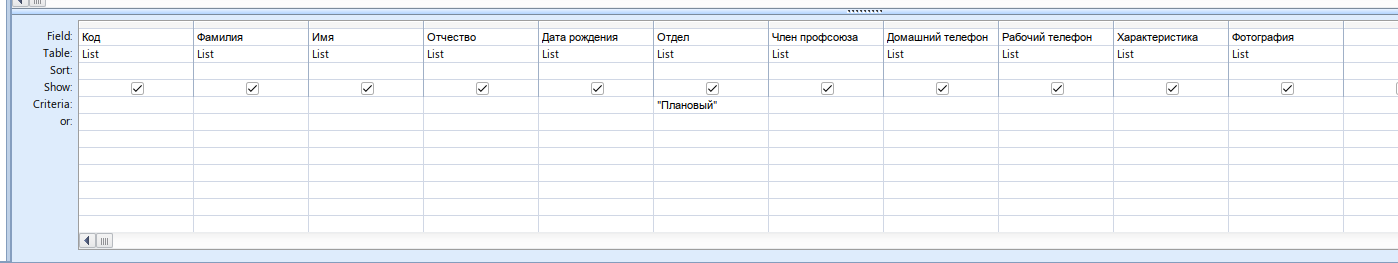


Рисунок 1.14

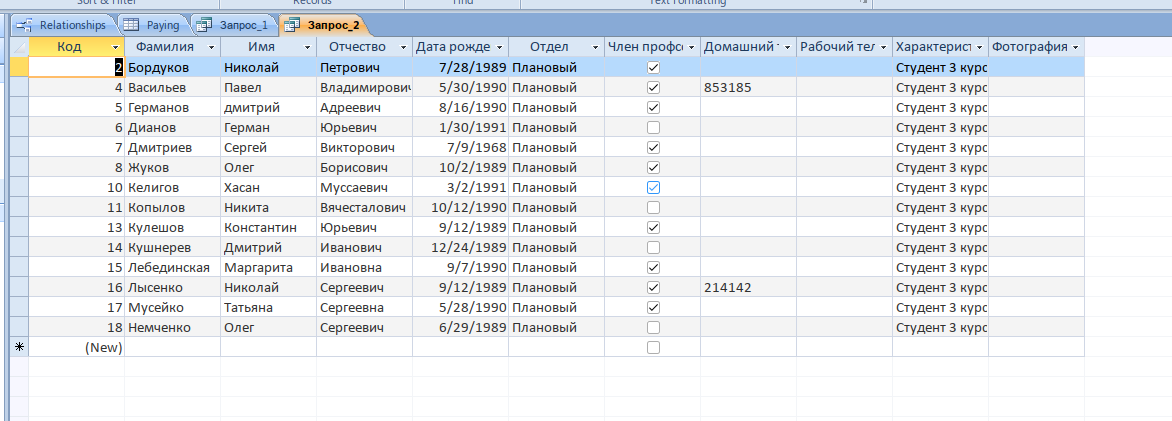


Рисунок 1.15

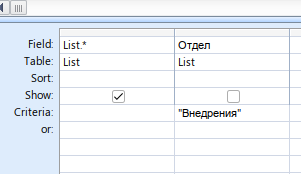


Рисунок 1.16

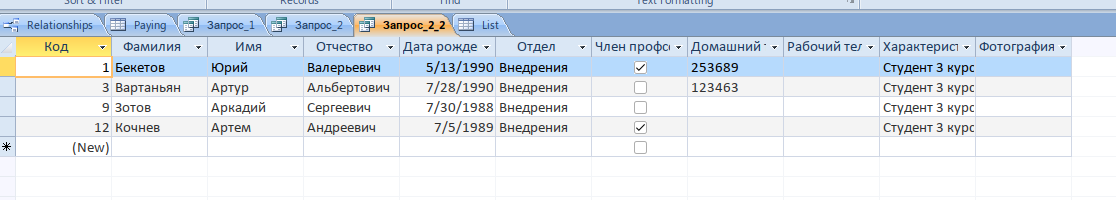


Рисунок 1.17

1. Создайте запрос по таблице Список и найдите записи с фамилиями начинающимися с буквы «А». (на рисунках 1.18-1.19)

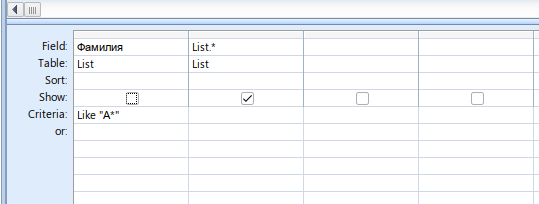


Рисунок 1.18

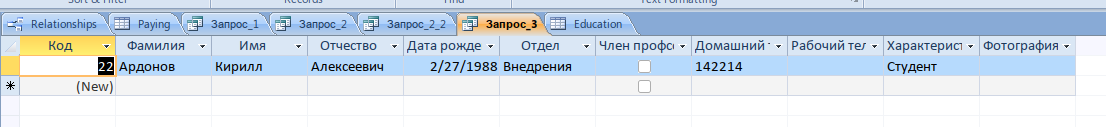


Рисунок 1.19

1. Создайте запрос по таблице Список и найдите записи сотрудников у которых есть домашний телефон. (на рисунках 1.20-1.21)

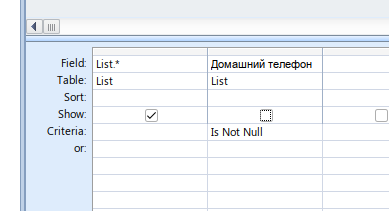


Рисунок 1.20

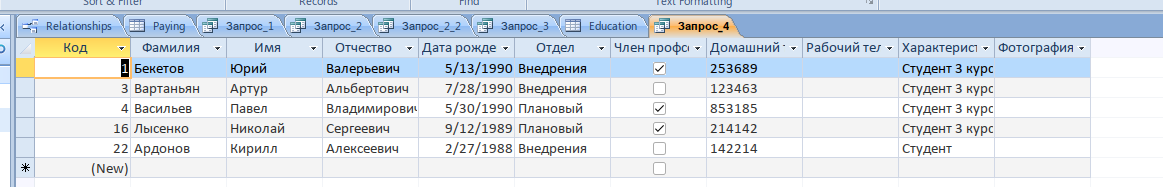


Рисунок 1.21

1. Создайте запрос по таблице Список и найдите записи сотрудников у которых нет домашнего телефона. (на рисунках 1.22-1.23)

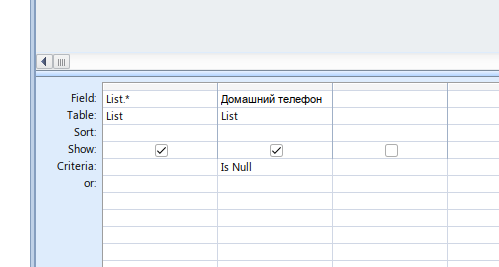


Рисунок 1.22

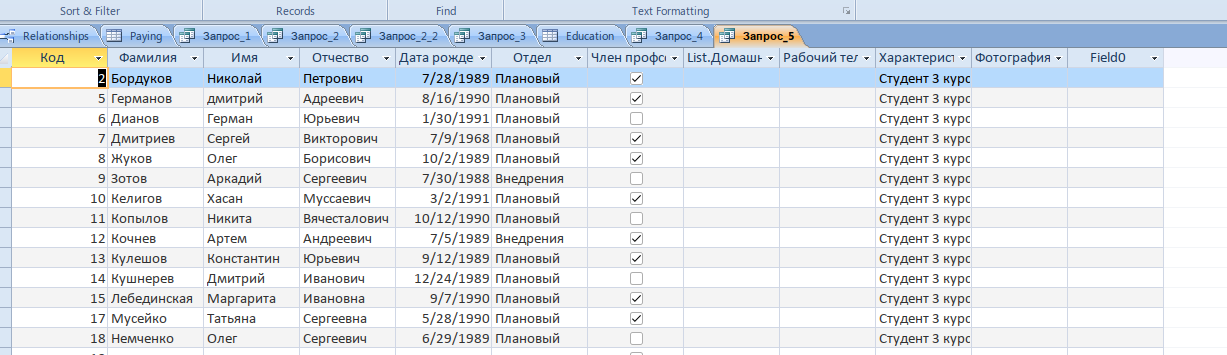


Рисунок 1.23

1. Создайте запрос по таблице Список и найдите записи сотрудников которые родились в апреле, затем 10 числа любого года и любого месяца . (на рисунках 1.24-1.27)

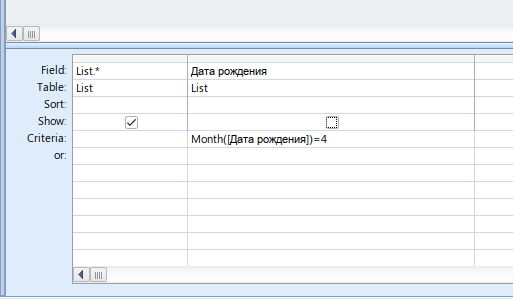


Рисунок 1.24

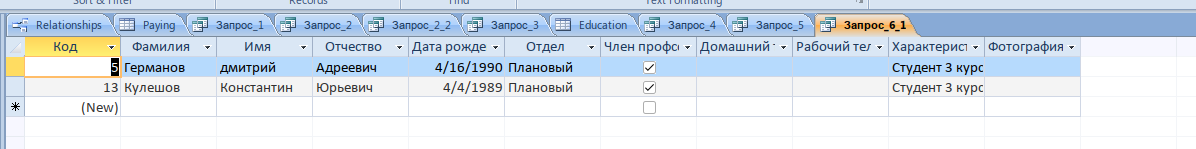


Рисунок 1.25

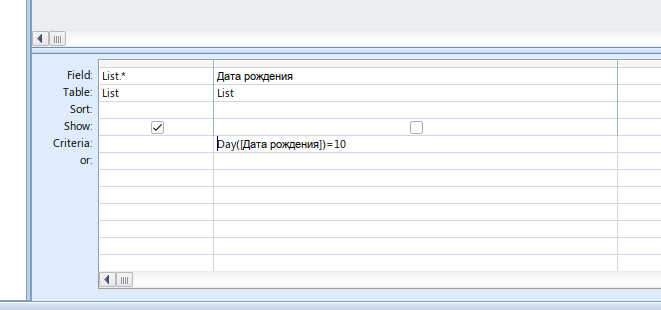


Рисунок 1.26

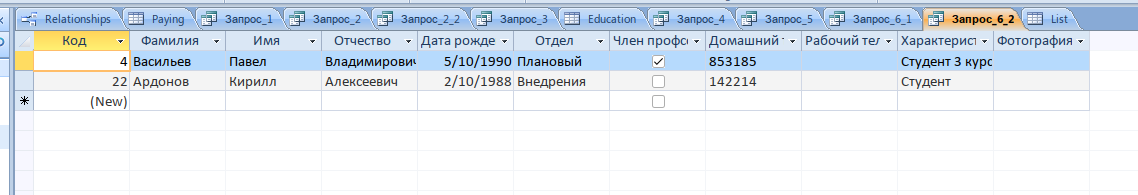


Рисунок 1.27

1. Создайте запрос по таблице Список и найдите записи сотрудников у которых код меньше 5 и они относятся к одному из отделов. (рисунок 1.28-1.29)

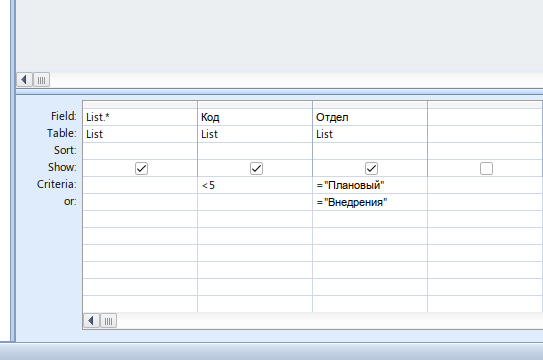


Рисунок 1.28

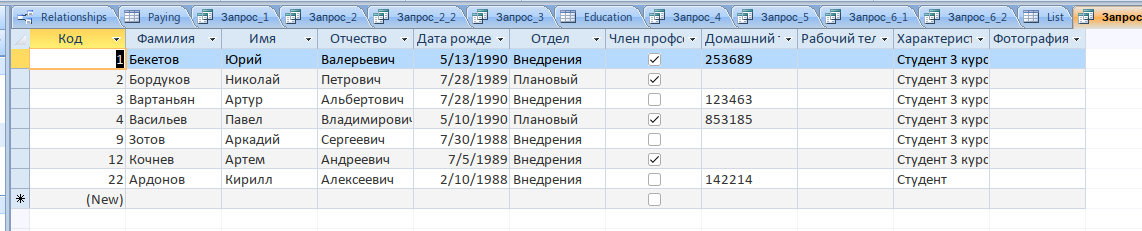


Рисунок 1.29

1. Создайте запрос по таблице Список и найдите записи сотрудников которые являются членами профсоюза. (рисунки 1.30-1.31)

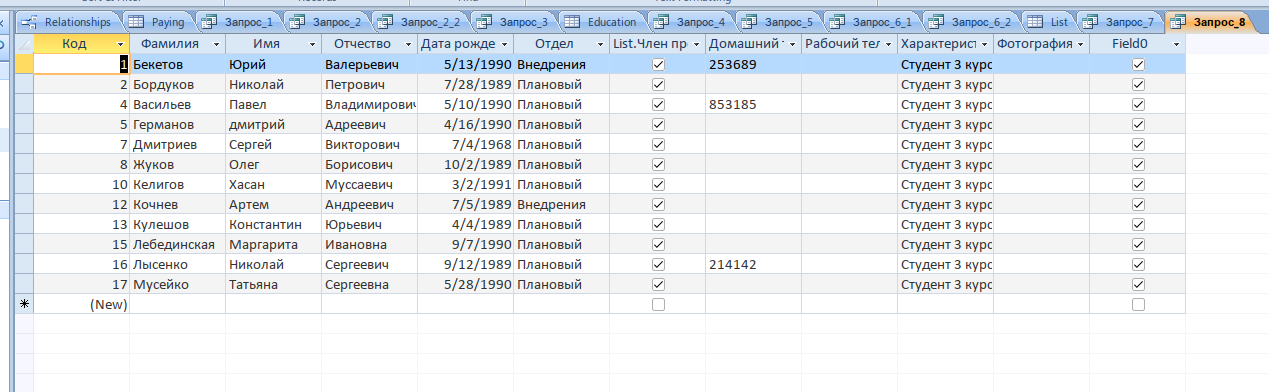


Рисунок 1.30

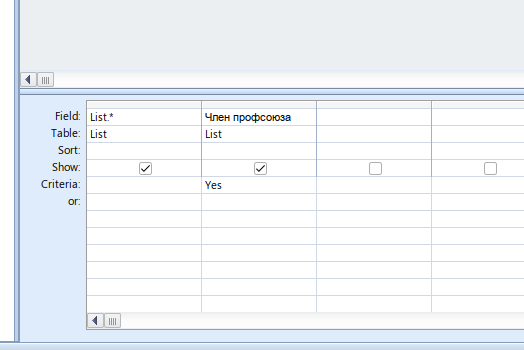


Рисунок 1.31

1. Создайте запрос по таблице список и таблице образование. Из таблицы список возьмите поля: Код, Отдел, фамилия, имя, отчество, Дата рождения. Из таблицы Образование возьмите все поля. (на рисунках 1.32-1.33)

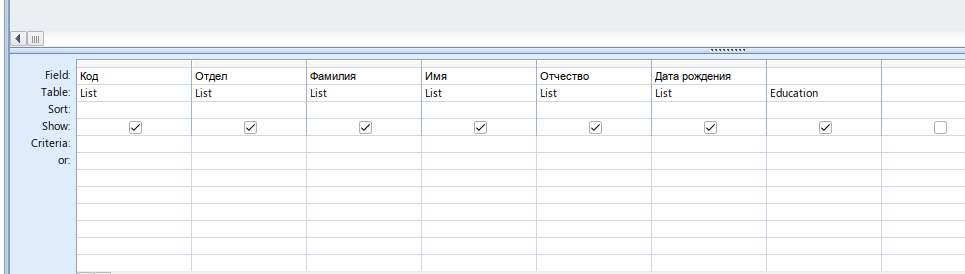


Рисунок 1.32

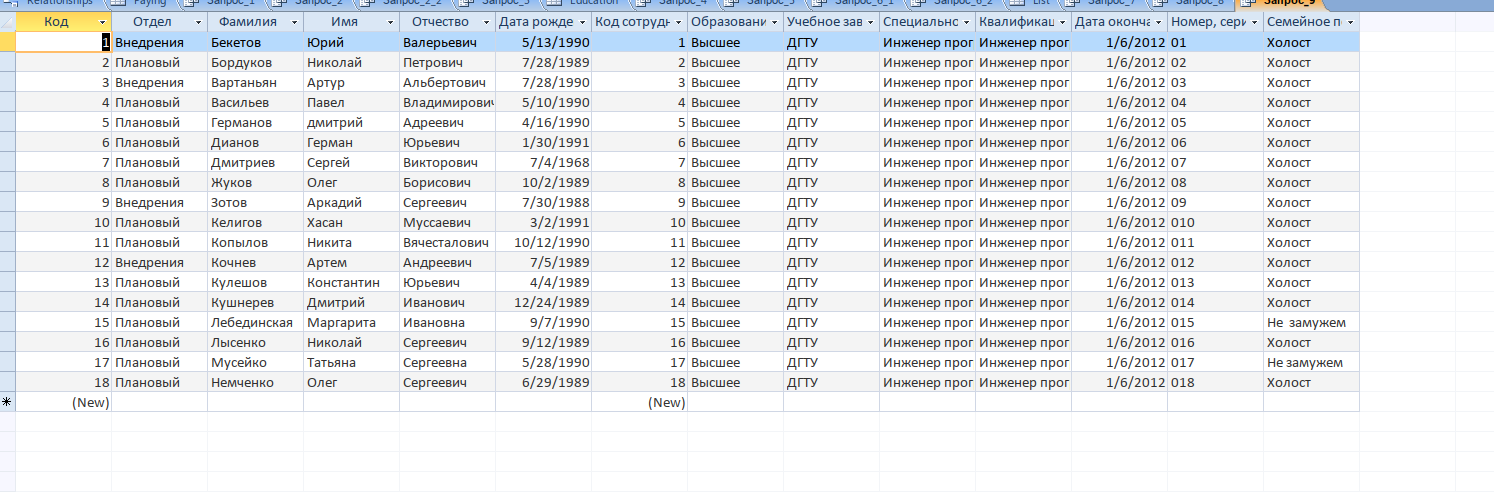


Рисунок 1.33

1. Создайте запрос по таблице список и таблице паспорт. Из таблицы список возьмите поля: Код, Отдел, фамилия, имя, отчество, Дата рождения.   
   Из таблицы Паспорт возьмите все поля. (на рисунках 1.34-1.35)

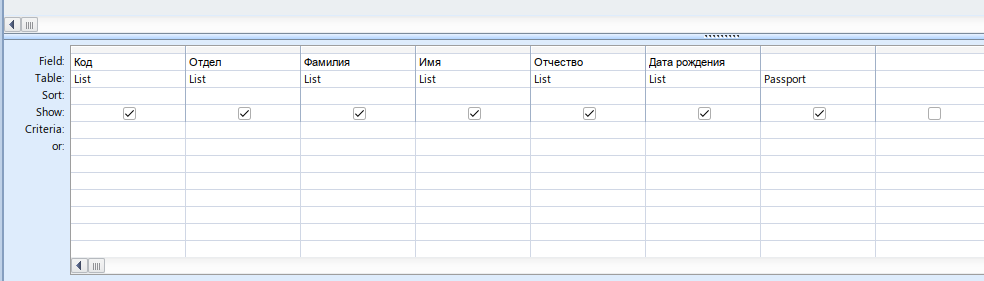


Рисунок 1.34

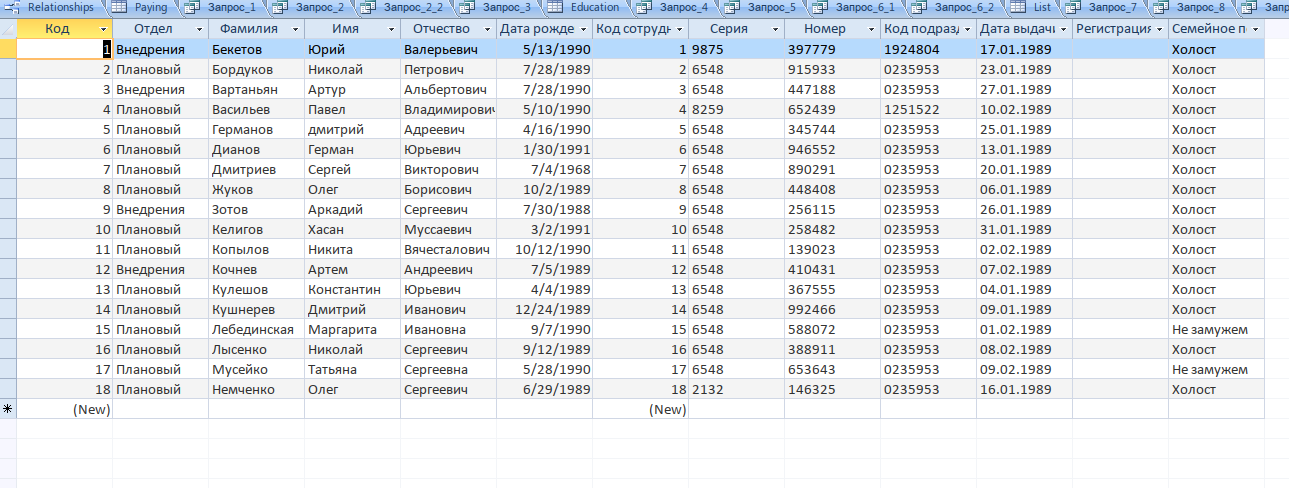


Рисунок 1.35

1. Создайте запрос по таблице список и таблице Дети. Из таблицы список возьмите поля: Код, Отдел, фамилия, имя, отчество, Дата рождения.   
   Из таблицы Дети возьмите все поля. (рисунки 1.36-1.37)

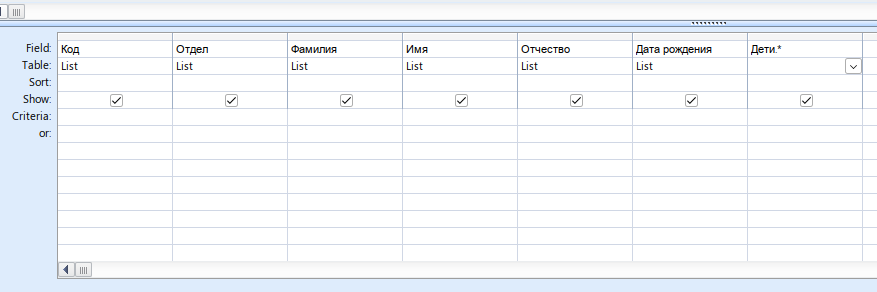


Рисунок 1.36

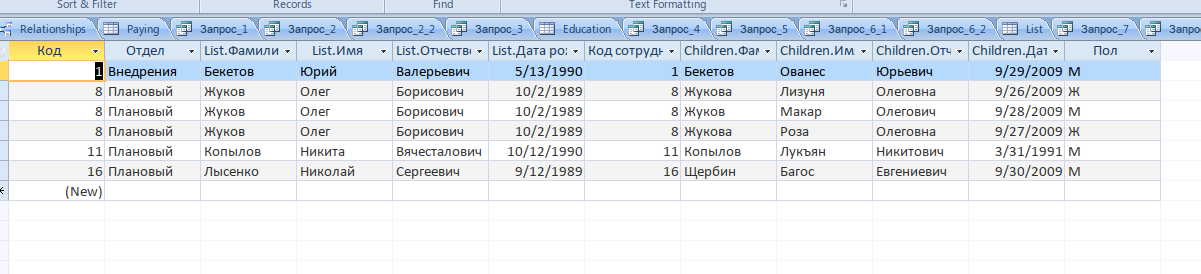


Рисунок 1.37

1. Создайте запрос по таблице список и таблице Оплата. Из таблицы список возьмите поля: Код, Отдел, фамилия, имя, отчество, Дата рождения.   
   Из таблицы Оплата возьмите все поля. (на рисунках 1.38-1.39)

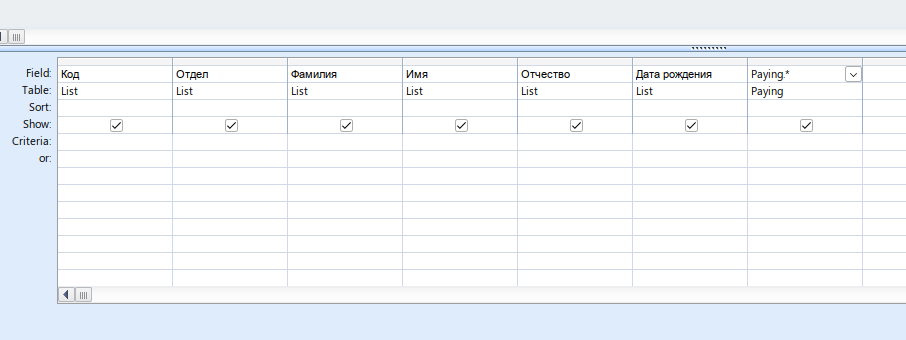


Рисунок 1.38

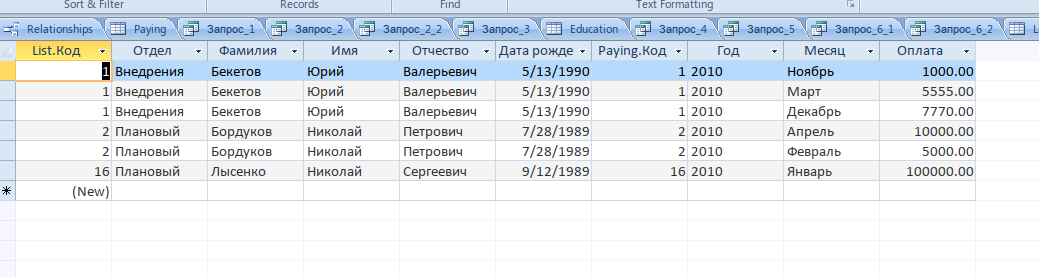


Рисунок 1.39

**Задание 4. Формы**

Форма по таблице «Список»

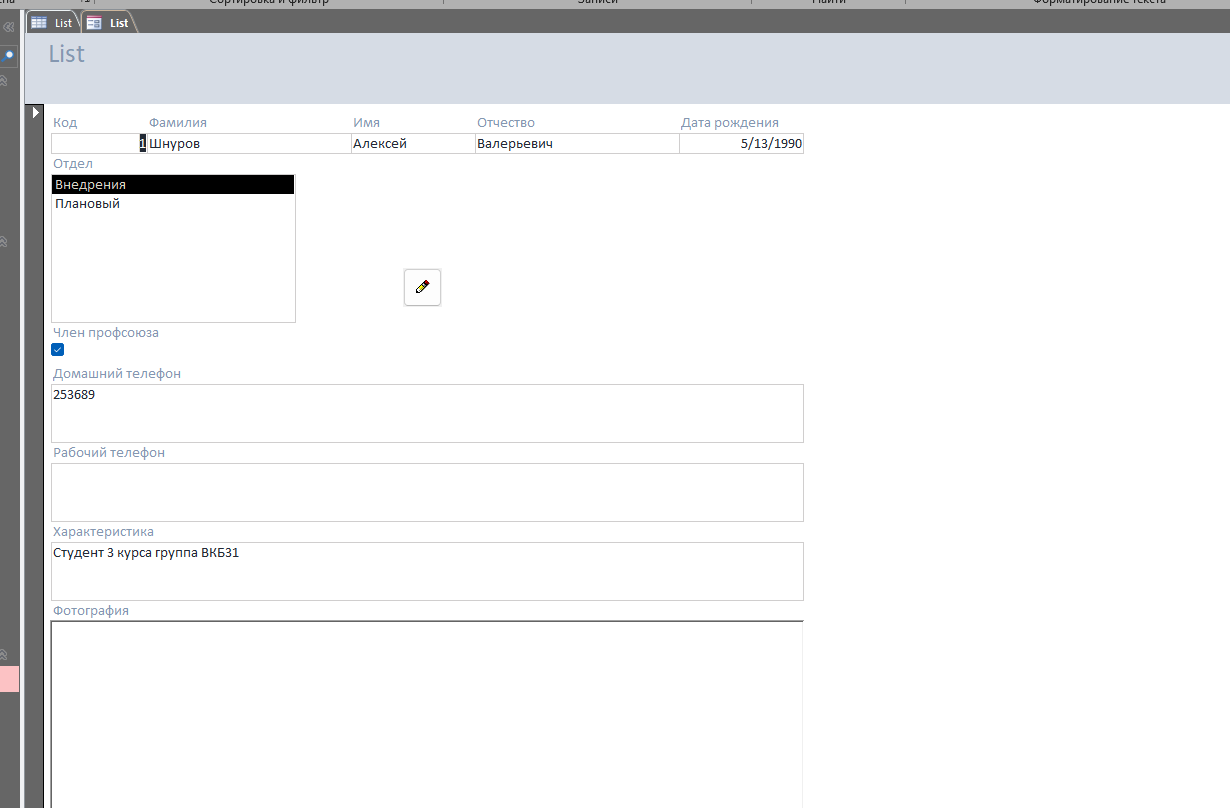


Рисунок 1.40

Форма по запросу «СписокОбразование»

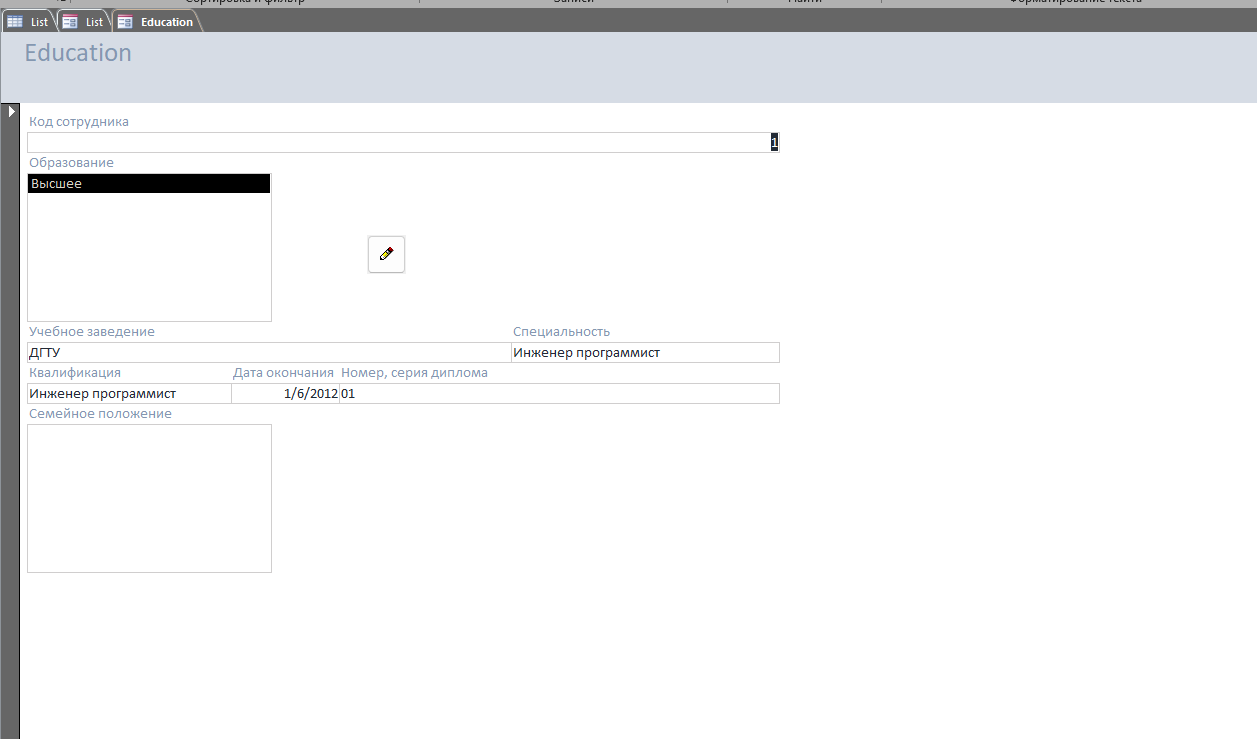


Рисунок 1.41

Форма по запросу «СписокПаспорт»

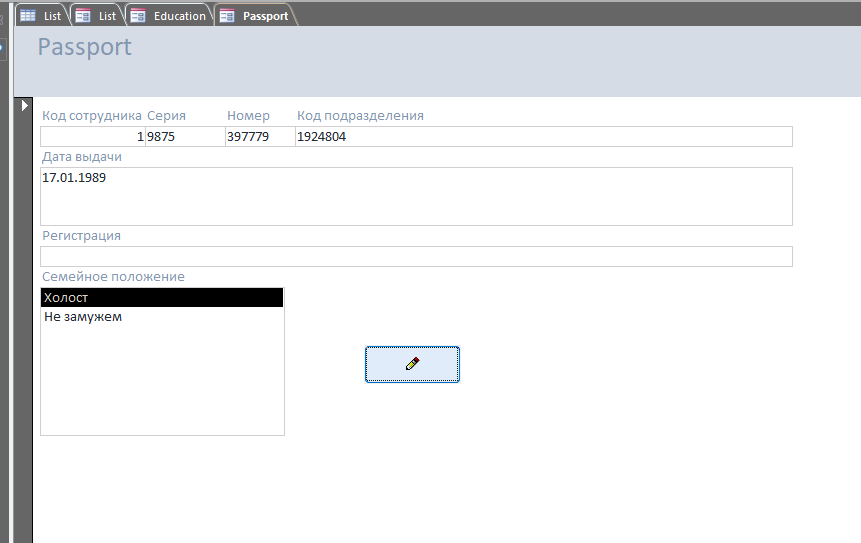


Рисунок 1.42

Составная форма по запросу «СписокОплата

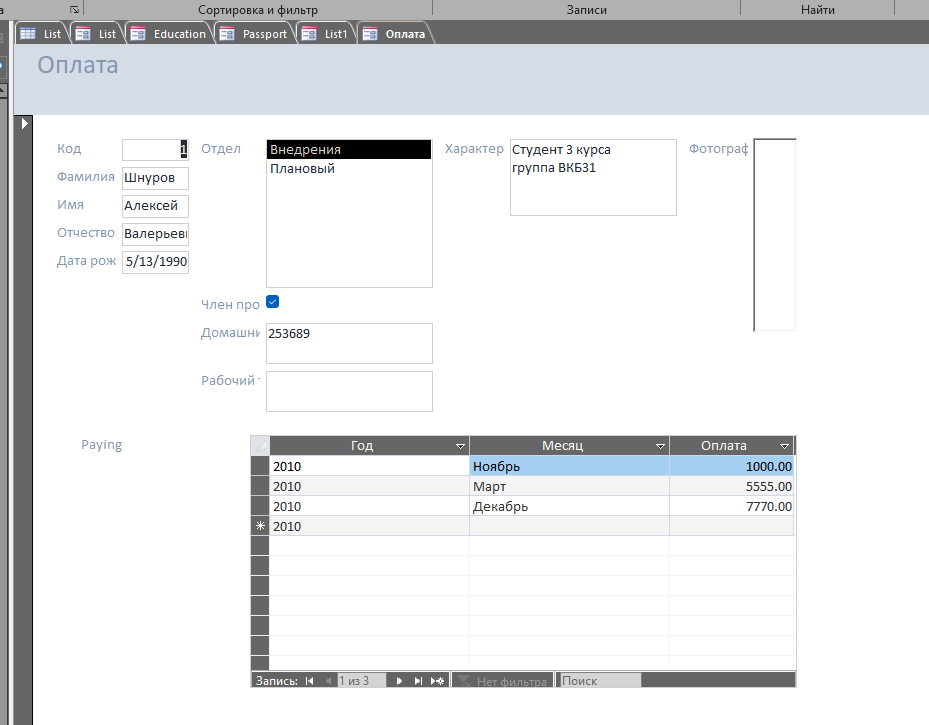


Рисунок 1.43

Составная форма по запросу «СписокДети»

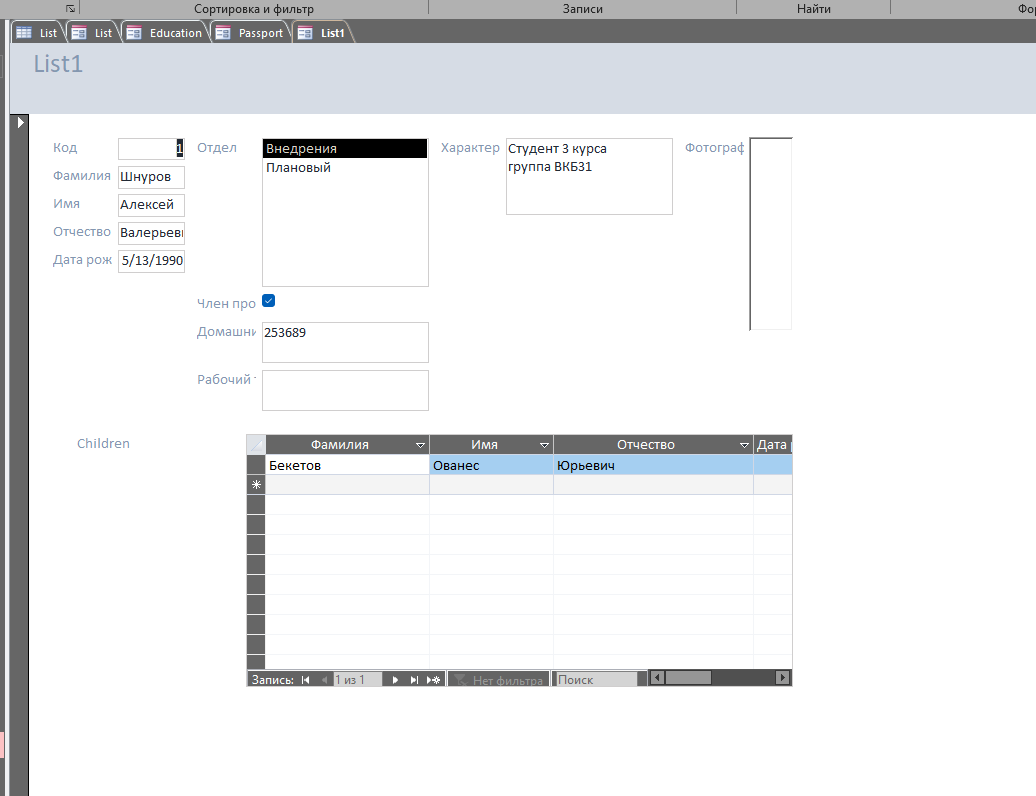


Рисунок 1.44

Задание 5. Создание и редактирование отчетов.

По запросу\_1

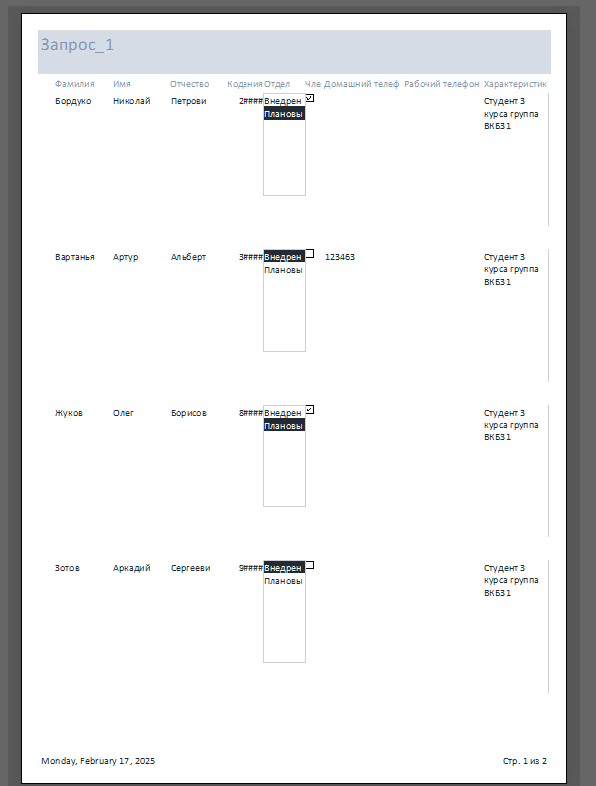


Рисунок 1.45

По запросу\_2

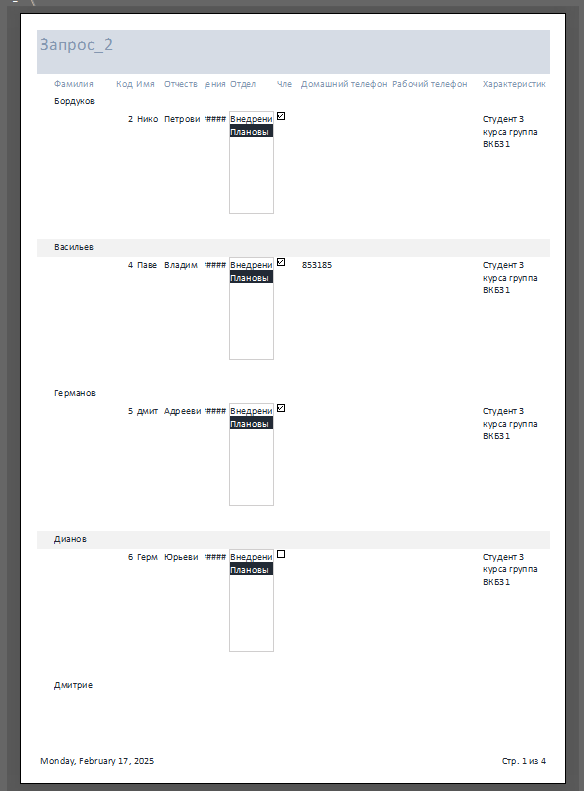


Рисунок 1.46

По запросу\_2\_2

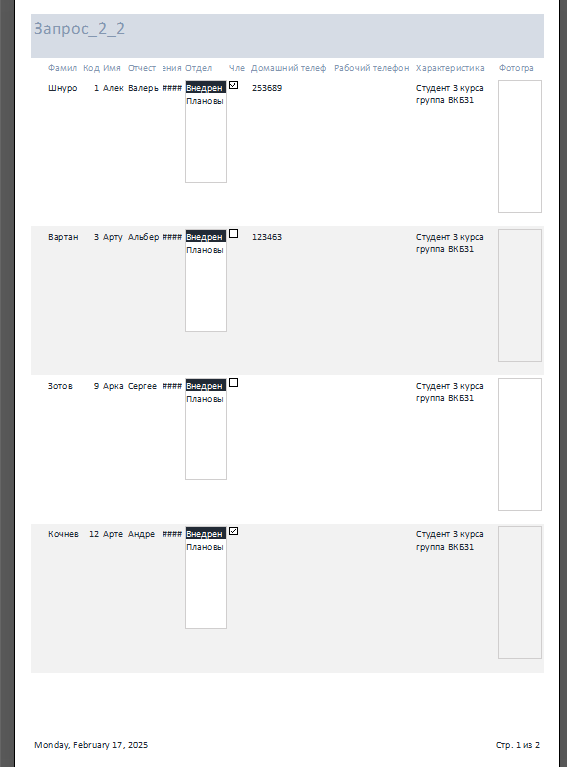


Рисунок 1.47

По запросу\_3

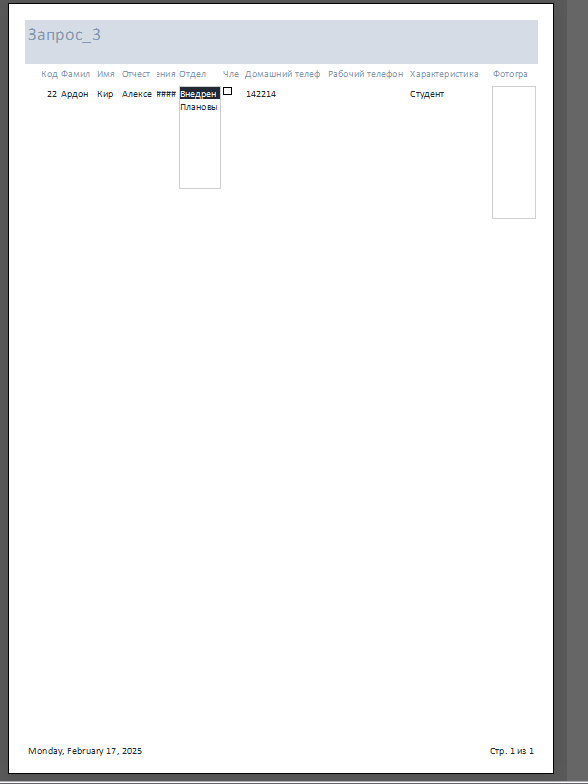


Рисунок 1.48

По запросу\_4

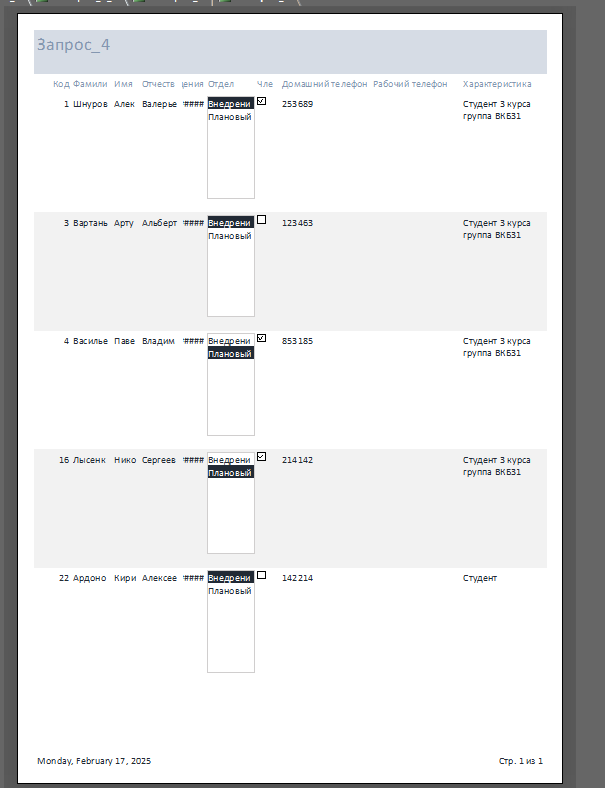


Рисунок 1.49

По запросу\_5

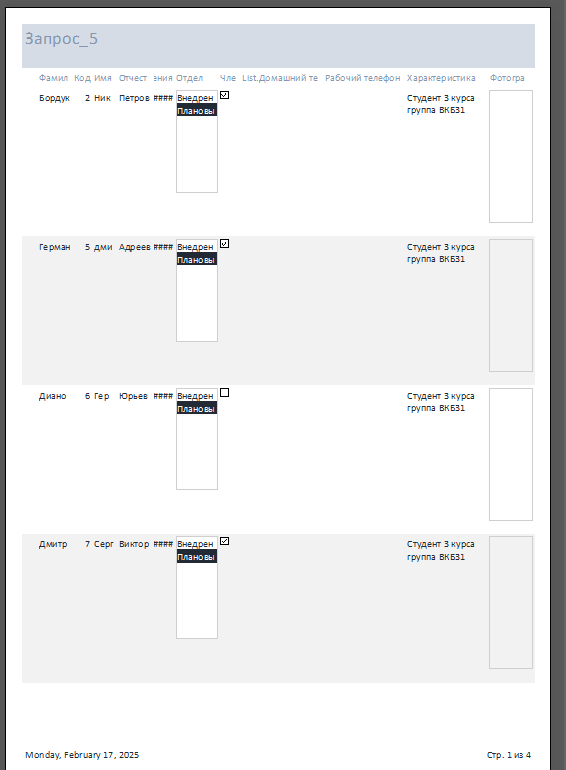


Рисунок 1.50

По запросу\_6\_1

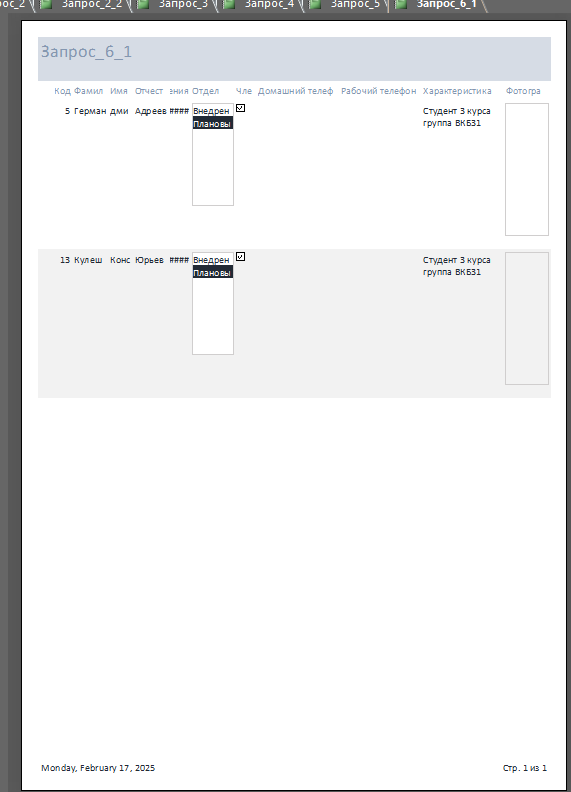


Рисунок 1.51

По запросу\_6\_2

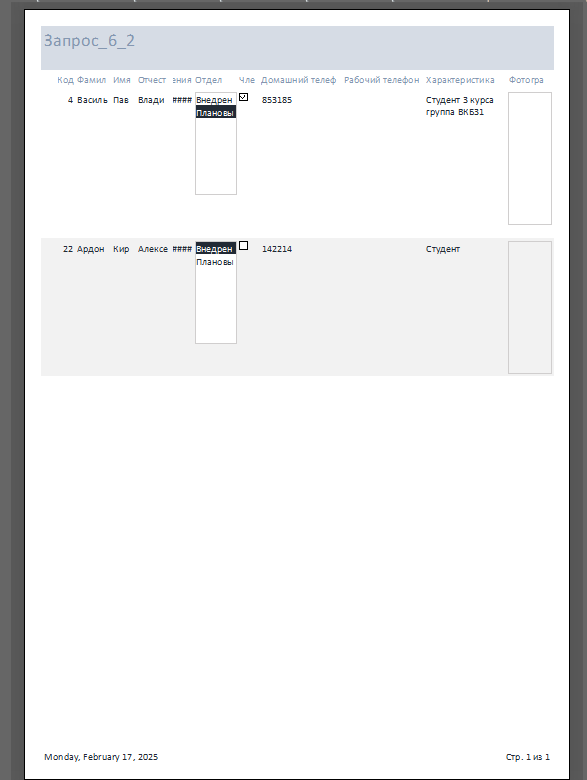


Рисунок 1.52

По запросу\_7

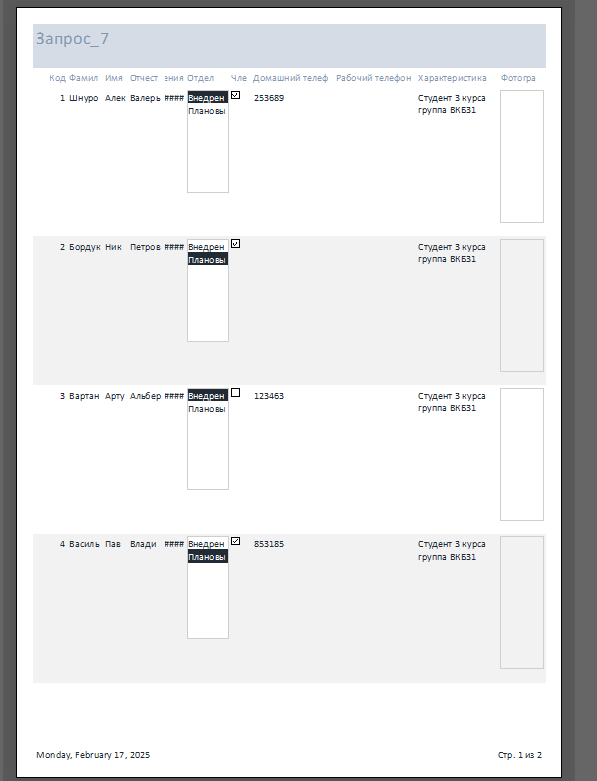


Рисунок 1.53

По запросу\_8

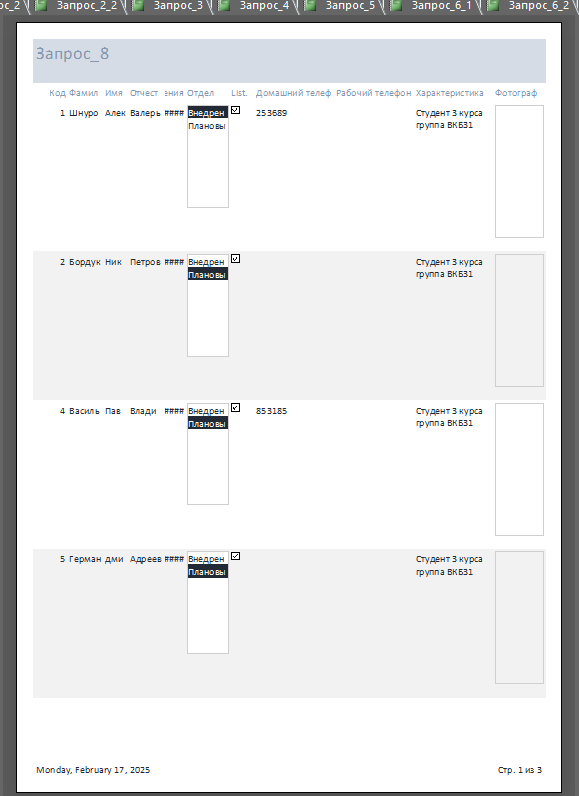


Рисунок 1.54

По запросу\_9

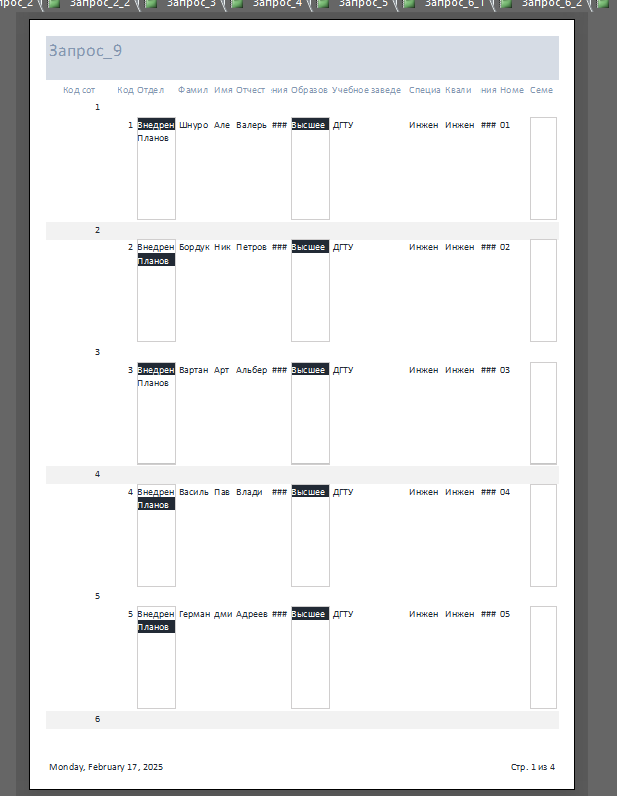


Рисунок 1.55

По запросу\_10

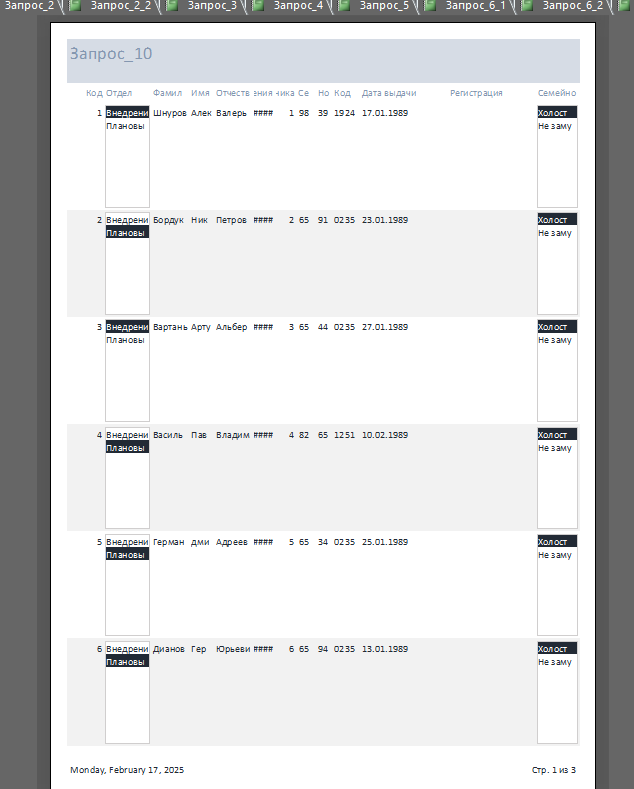


Рисунок 1.5

По запросу\_11

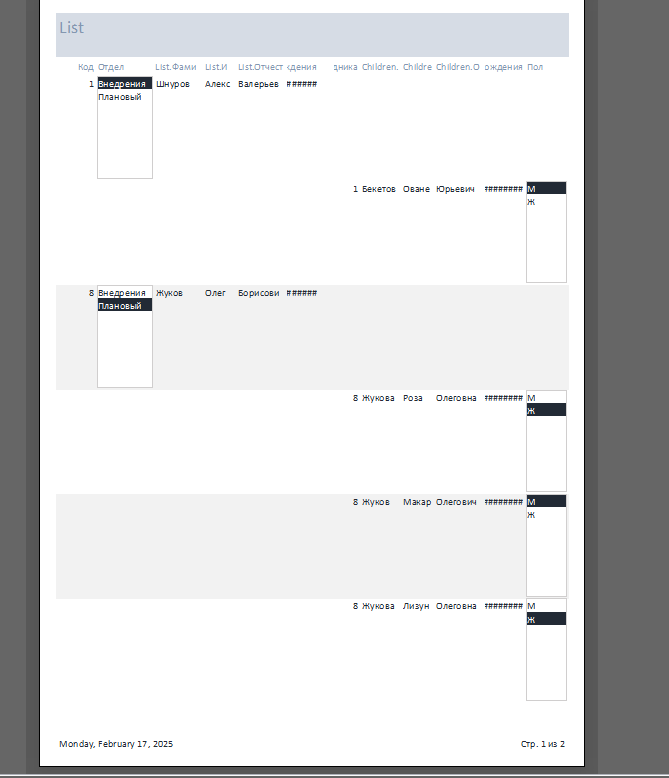


Рисунок 1.57

По запросу\_12

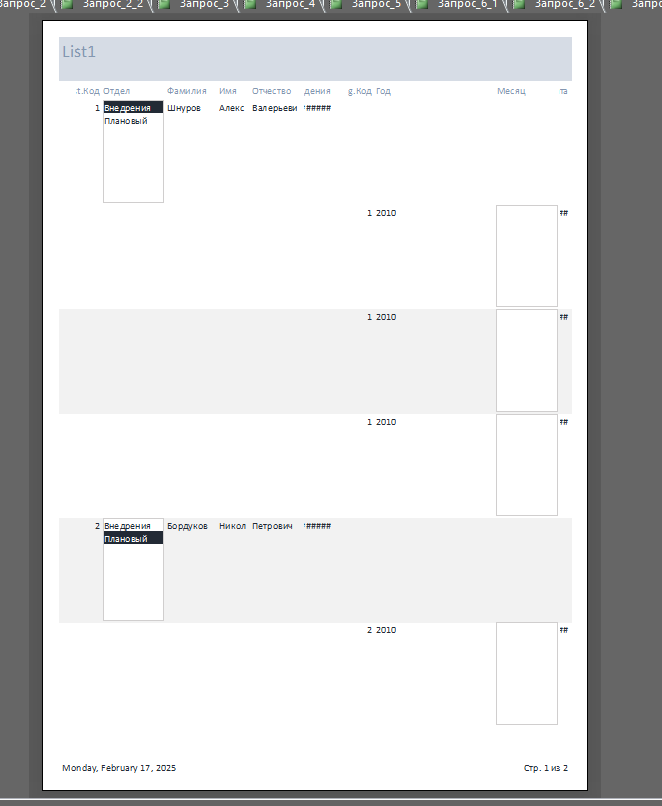


Рисунок 1.58

Лабораторная работа 2

Основы проектирования структуры баз данных

**Цель**: овладеть основными теоретическими знаниями и практическими приемами проектирования схемы реляционных отношений базы данных.

**Ход работы:**

В качестве предметной области выполнения лабораторной работы выбирается система управления библиотечным фондом города. С точки зрения проектирования базы данных в данной предметной области имеются следующие конечные пользователи: пользователи (читатели), сотрудники (библиотекари), администраторы системы. Опишем предметную область с точки зрения конечных пользователей.

Пользователи:

Это граждане, имеющие возможность посредством регистрации на ресурсе, пользоваться библиотечными фондами. Ожидается, что они будут пользоваться сервисом обмена печатной продукцией на безвозмеждной основе.

Сотрудники:

Это библиотекари, которые отвечают за учет пользователей, размещения и выдачи литературы, организацию работы в читальных залах и абонементах. Ожидается, что они обеспечат удобный интерфейс для учета выдачи и возврата литературы, возможность получения отчетов о работе.

Администраторы системы:

Системные администраторы: отвечают за техническую поддержку и развитие информационной системы. От них ожидается надежная и безопасная система с возможностью масштабирования, легкость в администрировании.

На базе представленных описаний можно выделить базовые объекты предметной области: читатель, библиотека, литература, сотрудники, система выдачи литературы

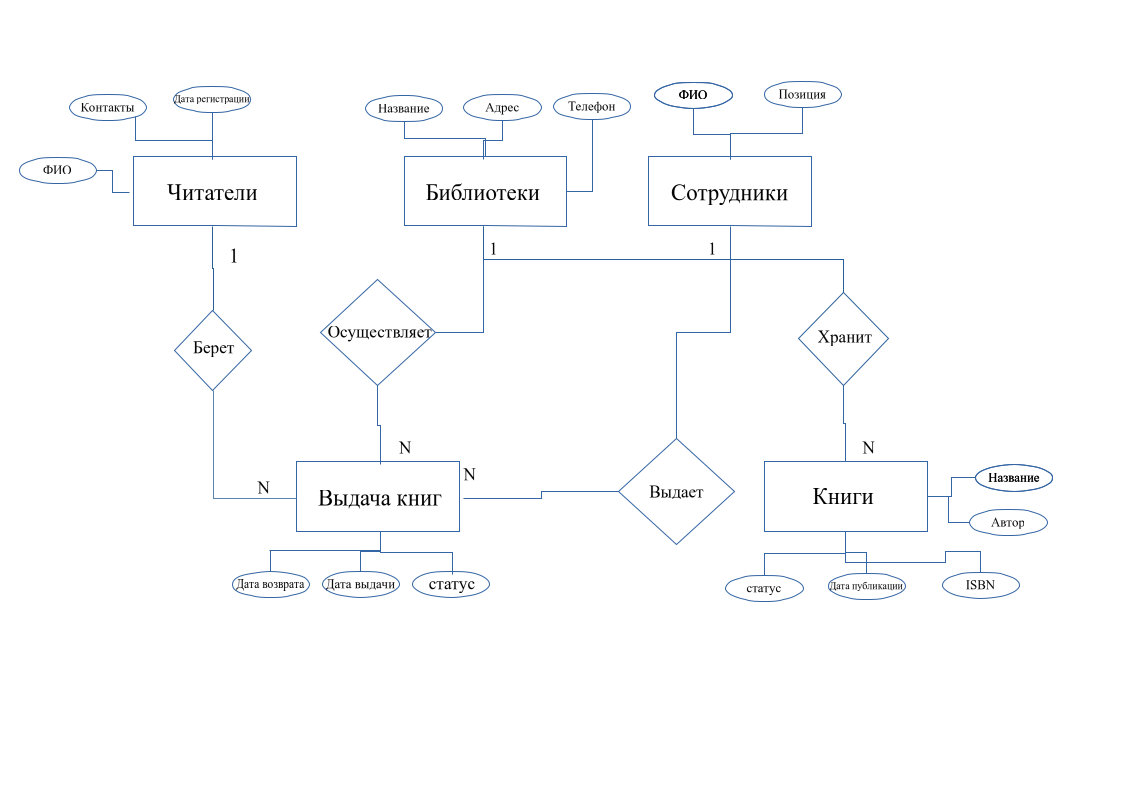


Рисунок 2.1 – ER диаграмма

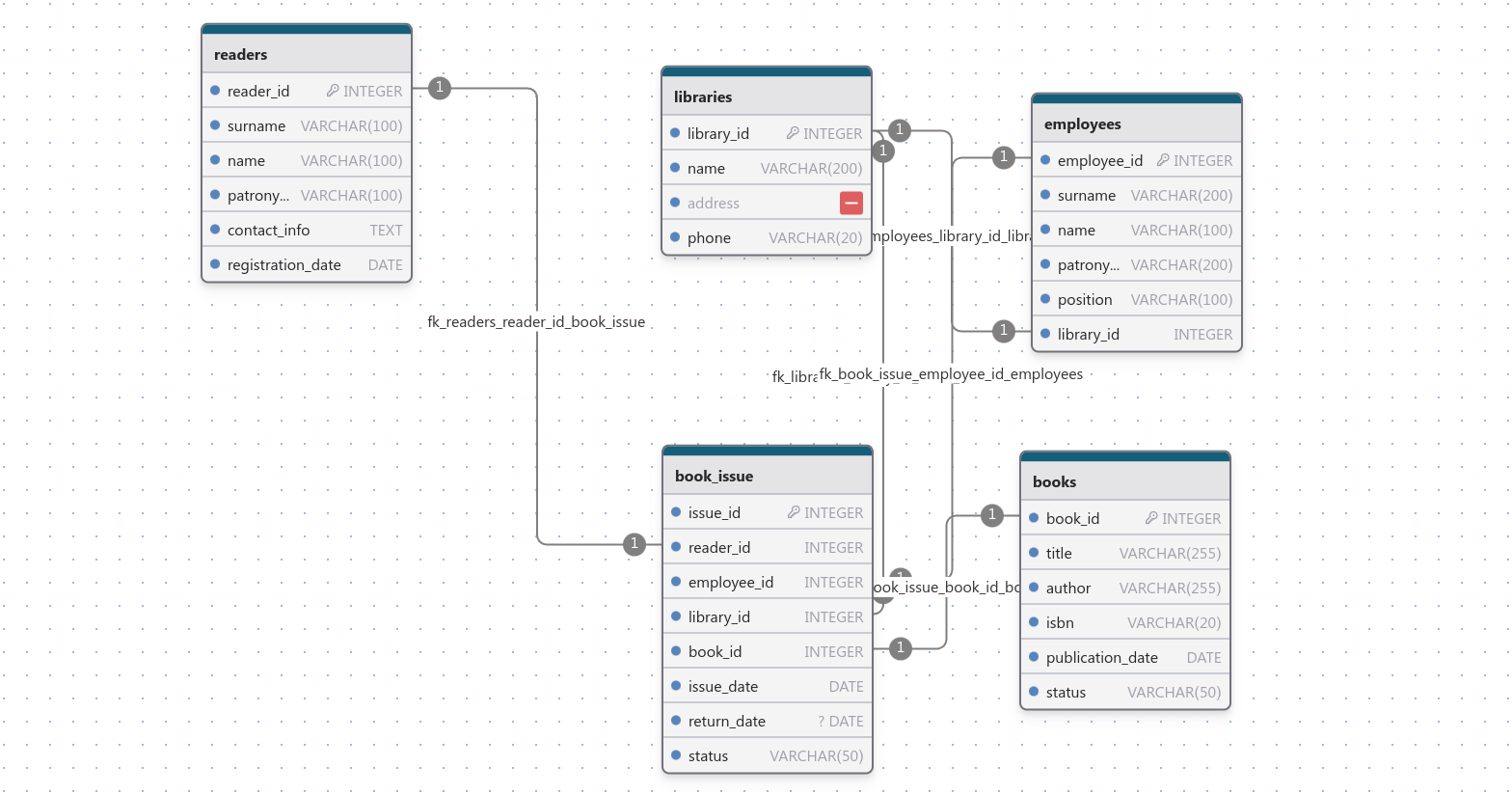


Рисунок 2.2 – Даталогическая диаграмма

Лабораторная работа №3

Основы работы с СУБД PostgreSQL

Цель: овладеть основными практическими навыками типового взаимодействия с использованием интергрированных утилит СУБД с БД на примере PostgreSQL

**2.1. Основные понятия реляционных баз данных**

**Упражнения**

1. Напишите запрос к таблице SUBJECT, выводящий для каждой ее строки идентификатор (номер) предмета обучения, его наименование, семестр, в котором он читается, и количество отводимых на него часов.

SELECT "SUBJ\_ID", "SUBJ\_NAME", "SEMESTR", "HOUR"

FROM "SUBJECT"

2. Напишите запрос, позволяющий вывести -все строки таблицы EXAM\_MARKS, в которых предмет обучения имеет номер (SUBJ\_ID), равный 12.

SELECT \*

FROM "EXAM\_MARKS"

WHERE "SUBJ\_ID" = 12

3. Напишите запрос, выбирающий все данные из таблицы STUDENT, расположив столбцы таблицы в следующем порядке: KURS, SURNAME, NАМЕ, STIPEND.

SELECT "KURS", "SURNAME", "NAME", "STIPEND"

FROM "STUDENT"

4. Напишите запрос SELECT, который для каждого предмета обучения (SUBJECT) выполняет вывод его наименования (SUBJ\_NAMЕ) и следом за ним количества часов (HOUR) в 4-м семестре (SEMESTR).

SELECT "SUBJ\_NAME", "HOUR", "SEMESTR"

FROM "SUBJECT"

WHERE "SEMESTR" = 4

5. Напишите запрос, позволяющий получить из таблицы EXAM\_MARKS значения столбца МАRК (экзаменационная оценка) для всех студентов, исключив из списка повторение одинаковых строк.

SELECT DISTINCT "STUDENT\_ID", "MARK"

FROM "EXAM\_MARKS"

6. Напишите запрос, который выполняет вывод списка фамилий студентов, обучающихся на третьем и более старших курсах.

SELECT "SURNAME", "KURS"

FROM "STUDENT"

WHERE "KURS" >=3

7. Напишите запрос, выбирающий данные фамилию, имя и номер курса для студентов, получающих стипендию больше 140.

SELECT "SURNAME", "NAME", "KURS"

FROM "STUDENT"

WHERE "STIPEND" > 140

8. Напишите запрос, выполняющий выборку нз таблицы SUBJECT названий всех предметов обучения, на которые отводится более 30 часов.

SELECT "SUBJ\_NAME"

FROM "SUBJECT"

WHERE "HOUR" > 30

9. Напишите запрос, который выполняет вывод.списка университетов, рейтинг которых превышает 300 баллов.

SELECT "UNIV\_NAME", "RATING"

FROM "UNIVERSITY"

WHERE "RATING" > 300

10. Напишите запрос к таблице STUDENT для вывода списка всех студентов со стипендией не меньше 100, живущих в Воронеже, с указанием фамилии (SURNAME), имени (NAME) и номера курса (KURS)

SELECT "SURNAME", "NAME", "KURS"

FROM "STUDENT"

WHERE "STIPEND" >=100 AND "CITY" = 'Воронеж'

11. Напишите запрос для получения списка студентов старше 25 лет, обучающихся на 1-м курсе.

SELECT \*

FROM "STUDENT"

WHERE "KURS" = 1 AND "BIRTHDAY" <= '1990-12-12'

12. Напишите запрос для получения списка предметов, для которых в 1-м семестре отведено более 100 часов.

SELECT "SUBJ\_NAME"

FROM "SUBJECT"

WHERE "SEMESTR" = 1 AND "HOUR" >= 100

13. Напишите запрос для получения списка преподавателей, живущих в Воронеже.

SELECT \*

FROM "LECTURER"

WHERE "CITY" = 'Воронеж'

14. Напишите запрос для получения списка университетов, расположенных в Москве и имеющих рейтинг меньший, чем у ВГУ. Константу в ограничении на рейтинг можно определить по этой же таблице.

SELECT \*

FROM "LECTURER"

WHERE "CITY" = 'Воронеж'

15. Напишите запрос для получения списка студентов, проживающих в Воронеже и не получающих стипендию.

SELECT \*

FROM "UNIVERSITY"

WHERE "CITY" = 'Москва' AND "RATING" > 296

16. Напишите запрос для получения списка студентов моложе 20 лет.

SELECT \*

FROM "STUDENT"

WHERE (extract(year from current\_date) - extract (year from("BIRTHDAY"))) > 30

17. Напишите запрос для получения списка студентов без определенного места жительства.

SELECT \*

FROM "STUDENT"

WHERE "CITY" IS NULL

**2.2 Операторы IN, BETWEEN, LIKE, IS NULL**

**Упражнения**

1. Напишите запрос, выполняющий вывод находящихся в таблице EXAM\_MARKS номеров предметов обучения, экзамены по которым сдавались между 10 и 20 января 2005 г.

SELECT "SUBJ\_ID"

FROM "EXAM\_MARKS"

WHERE "EXAM\_DATE" BETWEEN '2005-01-10' AND '2005-01-20'

2. Напишите запрос, выбирающий данные обо всех предметах обучения, экзамены по которым сданы студентами, имеющими идентификаторы 12 и 32.

SELECT \*

FROM "EXAM\_MARKS"

WHERE "STUDENT\_ID" IN (12, 32)

3. Напишите запрос, который выполняет вывод названий. предметов обучения, начинающихся на букву 'И'-;

SELECT "SUBJ\_NAME"

FROM "SUBJECT"

WHERE "SUBJ\_NAME" LIKE 'И%'

4. Напишите запрос, выбирающий сведения о студентах, у которых имена начинаются на букву 'И' или 'С'.

SELECT \*

FROM "STUDENT"

WHERE "NAME" LIKE 'И%' OR "NAME" LIKE 'С%'

5. Напишите запрос для выбора из таблицы EXAM\_MARKS записей, для которых отсутствуют значения оценок (поле MARK).

SELECT \*

FROM "EXAM\_MARKS"

WHERE "MARK" IS NULL

6. Напишите запрос, выполняющий вывод из таблицы EXAM\_MARKS записей, для которых в поле MARK проставлены значения оценок.

SELECT \*

FROM "EXAM\_MARKS"

WHERE "MARK" IS NOT NULL

7. Напишите запрос для получения списка преподавателей, проживающих в городах, в названиях которых присутствует дефис.

SELECT \*

FROM "LECTURER"

WHERE "CITY" LIKE '%-%'

8. Напишите запрос для получения списка предметов, названия которых оканчиваются на 'ия'.

SELECT \*

FROM "SUBJECT"

WHERE "SUBJ\_NAME" LIKE '%ия'

9. Напишите запрос для получения списка студентов, фамилии которых начинаются на 'Ков' или на 'Куз'.

SELECT \*

FROM "STUDENT"

WHERE "SURNAME" LIKE 'Ков%' OR "SURNAME" LIKE 'Куз%'

10. Напишите запрос для получения списка предметов обучения, названия которых состоят из более одного слова.

SELECT \*

FROM "SUBJECT"

WHERE "SUBJ\_NAME" LIKE '% %';

11. Напишите запрос для получения списка учебных заведений, аббревиатура которых состоит как минимум из 4 букв.

SELECT \*

FROM "UNIVERSITY"

WHERE "UNIV\_NAME" LIKE '\_\_\_\_%'

12. Напишите запрос для получения списка студентов, фамилии которых состоят из трех букв.

SELECT \*

FROM "STUDENT"

WHERE "SURNAME" LIKE '\_\_\_'

**2.3 Преобразование вывода и встроенные функции**

**Упражнения**

1. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала один столбец, содержащий последовательность разделенных символом ";" (точка с запятой) значений всех столбцов этой таблицы; при этом текстовые значения должны отображаться прописными символами (верхний регистр), т. е. быть представленными в следующем виде:

10; КУЗНЕЦОВ; БОРИС; 0; БРЯНСК; 8.12.1987; 10.

SELECT UPPER("STUDENT\_ID"||';'||"SURNAME"||';'||"NAME"||';'||"KURS"||';'||"CITY"||';'||"BIRTHDAY"||';'||"UNIV\_ID")

FROM "STUDENT"

2. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: Б. КУЗНЕЦОВ; место жительства — БРЯНСК; родился — 8.12.87.

SELECT SUBSTR("NAME", 1,1)||'.'||UPPER("SURNAME")||';'||' место жительства'||' -- '||UPPER("CITY")||';'||' родился'||' -- '||

TO\_CHAR("BIRTHDAY",'DD.MM.YY')

FROM "STUDENT"

3. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: б. кузнецов; место жительства — брянск; родился: 8-дек-1987.

SELECT LOWER(SUBSTR("NAME", 1,1))||'.'||LOWER("SURNAME")||';'||' место жительства'||' -- '||LOWER("CITY")||';'||' родился'||': '||

TO\_CHAR("BIRTHDAY",'DD.Mon.YYYY')

FROM "STUDENT"

4. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: Борис Кузнецов родился в 1987 году.

SELECT "NAME"||' '||"SURNAME"||' '|| 'родился в' ||' '|| TO\_CHAR("BIRTHDAY",'YYYY') ||' '||'году'

FROM "STUDENT"

5. Составьте запрос, выводящий фамилии, имена студентов и величину получаемых ими стипендий, при этом значения стипендий должны быть увеличены в 100 раз.

SELECT "SURNAME", "NAME", "STIPEND"\*100

FROM "STUDENT"

6. То же, что и в упр. 4, но только для студентов 1, 2 и 4 курсов и таким образом, чтобы фамилии и имена были выведены прописными буквами.

SELECT UPPER("NAME"||' '||"SURNAME")||' '|| 'родился в' ||' '|| TO\_CHAR("BIRTHDAY",'YYYY') ||' '||'году'

FROM "STUDENT"

WHERE "KURS" IN (1,2,4)

7 Составьте запрос для таблицы UNIVERSITY таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: Код-10; ВГУ-r. ВОРОНЕЖ; Рейтинг=296.

SELECT 'Код'||'-'||"UNIV\_ID"||'; '||"UNIV\_NAME"||'-г. '||UPPER("CITY")||'; '||'Рейтинг'||'='||"RATING"

FROM "UNIVERSITY"

8. То же; что и в упр. 7, но значения рейтинга требуется округлить до первого знака (например, значение -382 округляется-до 400).

SELECT 'Код'||'-'||"UNIV\_ID"||'; '||"UNIV\_NAME"||'-г. '||UPPER("CITY")||'; '||'Рейтинг'||'='||ROUND("RATING",-2)

FROM "UNIVERSITY"

**2.4 Агрегирование и групповые функции**

**Упражнения**

1. Напишите запрос для подсчета количества студентов, сдавших (поле MARK непустое) экзамен по предмету обучения с идентификатором 12.

SELECT COUNT(\*)

FROM "EXAM\_MARKS"

WHERE "SUBJ\_ID" = 12 AND "MARK" IS NOT NULL;

2. Напишите запрос, который позволяет подсчитать в таблице EXAM\_MARKS количество различных предметов обучения (использовать оператор DISTINCT.

SELECT COUNT(DISTINCT "SUBJ\_ID")

FROM "EXAM\_MARKS";

3. Напишите запрос, который для каждого студента выполняет выборку его идентификатора и минимальной из полученных им оценок.

SELECT "STUDENT\_ID", MIN("MARK")

FROM "EXAM\_MARKS"

GROUP BY "STUDENT\_ID";

4. Напишите запрос, который для каждого студента выполняет выборку его идентификатора и максимальной из полученных им оценок.

SELECT "STUDENT\_ID", MAX("MARK")

FROM "EXAM\_MARKS"

GROUP BY "STUDENT\_ID";

6. Напишите запрос, который для каждого конкретного дня сдачи экзамена выводит данные о количестве студентов, сдававших экзамен в этот день.

SELECT "EXAM\_DATE", COUNT("STUDENT\_ID")

FROM "EXAM\_MARKS"

GROUP BY "EXAM\_DATE";

7. Напишите запрос, выдающий средний балл для каждого студента.

SELECT AVG("MARK")

FROM "EXAM\_MARKS"

GROUP BY "STUDENT\_ID"

8. Напишите запрос, выдающий средний балл для каждого экзамена.

SELECT AVG("MARK")

FROM "EXAM\_MARKS"

GROUP BY "SUBJ\_ID"

9. Напишите запрос, определяющий количество сдававших студентов для каждого экзамена.

SELECT COUNT(DISTINCT"STUDENT\_ID")

FROM "EXAM\_MARKS"

GROUP BY "SUBJ\_ID"

10. Напишите запрос для определения количества предметов, изучаемых на каждом курсе.

SELECT COUNT(DISTINCT "SUBJ\_ID")

FROM "EXAM\_MARKS", "STUDENT"

GROUP BY "KURS"

11. Для каждого университета напишите запрос, выводящий суммарную стипендию обучающихся в нем студентов, с последующей сортировкой списка по этому значению.

SELECT "UNIV\_ID", SUM("STIPEND")

FROM "STUDENT"

GROUP BY "UNIV\_ID"

12. Для каждого семестра напишите запрос, выводящий общее количество часов, отводимое на изучение соответствующих предметов.

SELECT "SEMESTR", SUM("HOUR")

FROM "SUBJECT"

GROUP BY "SEMESTR"

13. Для каждого студента напишите запрос, выводящий среднее значение оценок, полученных им по каждому предмету.

SELECT AVG("MARK")

FROM "EXAM\_MARKS"

GROUP BY "SUBJ\_ID", "STUDENT\_ID"

14. Напишите запрос, выводящий количество студентов, проживающих в каждом городе. Список отсортировать в порядке убывания количества студентов.

SELECT COUNT(DISTINCT "STUDENT\_ID")

FROM "STUDENT"

ORDER BY COUNT(DISTINCT "STUDENT\_ID") DESC

15. Для каждого университета напишите запрос, выводящий количество обучающихся в нем студентов, с последующей сортировкой списка по этому количеству.

SELECT COUNT("STUDENT\_ID"), "UNIV\_ID"

FROM "STUDENT"

GROUP BY "UNIV\_ID"

16. Для каждого университета напишите запрос, выводящий количество работающих в нем преподавателей, с последующей сортировкой списка по этому количеству.

SELECT COUNT("LECTURER\_ID"), "UNIV\_ID"

FROM "LECTURER"

GROUP BY "UNIV\_ID"

ORDER BY COUNT("LECTURER\_ID") DESC

17. Для каждого университета напишите запрос, выводящий сумму стипендии, выплачиваемой студентам каждого курса.

SELECT SUM("STIPEND"), "UNIV\_ID", "KURS"

FROM "STUDENT"

GROUP BY "UNIV\_ID", "KURS"

ORDER BY "UNIV\_ID", "KURS"

18. Для каждого города напишите запрос, выводящий максимальный рейтинг университетов, в нем расположенных, с последующей сортировкой списка по значениям рейтингов.

SELECT MAX("RATING"), "CITY"

FROM "UNIVERSITY"

GROUP BY "CITY"

19. Для каждого дня сдачи экзаменов напишите запрос, выводящий среднее значение всех экзаменационных оценок.

SELECT AVG("MARK"), "EXAM\_DATE"

FROM "EXAM\_MARKS"

GROUP BY "EXAM\_DATE"

20. Для каждого дня сдачи экзаменов напишите запрос, выводящий максимальные оценки, полученные по каждому предмету.

SELECT MAX("MARK"), "EXAM\_DATE"

FROM "EXAM\_MARKS"

GROUP BY "EXAM\_DATE"

21. Для каждого дня сдачи экзаменов напишите запрос, выводящий общее количество экзаменов, сдававшихся каждым студентом.

SELECT COUNT("SUBJ\_ID"), "EXAM\_DATE"

FROM "EXAM\_MARKS"

GROUP BY "EXAM\_DATE"

22. Для каждого преподавателя напишите запрос, выводящий количество преподаваемых им предметов.

SELECT COUNT("SUBJ\_ID"), "LECTURER\_ID"

FROM "SUBJ\_LECT"

GROUP BY "SUBJ\_ID", "LECTURER\_ID"

23. Для каждого предмета напишите запрос, выводящий количество преподавателей, ведущих по нему занятия

SELECT COUNT("LECTURER\_ID"), "SUBJ\_ID"

FROM "SUBJ\_LECT"

GROUP BY "SUBJ\_ID"

24. Напишите запрос, выполняющий вывод количества студентов, имеющих только отличные оценки.

SELECT COUNT("STUDENT\_ID")

FROM "EXAM\_MARKS"

WHERE "MARK" = 5

GROUP BY "MARK"

25. Напишите запрос, выполняющий вывод количества экзаменов; сданных (с положительной оценкой) студентом с идентификатором 32.

SELECT COUNT("SUBJ\_ID")

FROM "EXAM\_MARKS"

WHERE "MARK">=3 AND "STUDENT\_ID" = 32

**2.5.1 Вложенные подзапросы**

**Упражнения**

1. Напишите запрос, выводящий список студентов, получающих максимальную стипендию, отсортировав его в алфавитном порядке по фамилиям.

SELECT \*

FROM "STUDENT"

WHERE "STIPEND" =

(SELECT MAX("STIPEND")

FROM "STUDENT"

)

ORDER BY "SURNAME" ASC

2. Напишите запрос, выводящий список студентов, получающих стипендию, превышающую среднее значение стипендии.

SELECT \*

FROM "STUDENT"

WHERE "STIPEND">

(SELECT AVG("STIPEND")

FROM "STUDENT")

8. Напишите запрос, выводящий список, студентов, обучающихся в Воронеже, с последующей сортировкой по идентификаторам университетов и курсам.

SELECT \*

FROM "STUDENT"

WHERE "UNIV\_ID" IN

(SELECT "UNIV\_ID"

FROM "UNIVERSITY"

WHERE "CITY" = 'Воронеж')

ORDER BY "UNIV\_ID", "KURS"

9. Напишите запрос, выводящий список предметов, на изучение которых отведено максимальное количество часов.

SELECT \*

FROM "SUBJECT"

WHERE "HOUR" IN

(SELECT MAX("HOUR")

FROM "SUBJECT")

10. Напишите запрос, выполняющий вывод имен и фамилий студентов, место проживания которых не совпадает с городом, в котором находится их университет.

SELECT "NAME", "SURNAME"

FROM "STUDENT", "UNIVERSITY"

WHERE "STUDENT"."UNIV\_ID" = "UNIVERSITY"."UNIV\_ID" and "STUDENT"."CITY" <> "UNIVERSITY"."CITY"

11. Напишите запрос, выводящий список университетов, расположенных в Москве и имеющих рейтинг меньший, чем у ВГУ.

SELECT \*

FROM "UNIVERSITY"

WHERE "CITY" = 'Москва' AND "RATING" >

(SELECT "RATING"

FROM "UNIVERSITY"

WHERE "UNIV\_NAME" = 'ВГУ')

**2.5.2 Операции соединения таблиц посредством ссылочной целостности.**

**Упражнения**

1:. Напишите запрос для получения списка университетов с указанием количества студентов, обучающихся на каждом курсе.

SELECT "UNIV\_NAME", "KURS" ,COUNT("STUDENT\_ID")

FROM "UNIVERSITY" JOIN "STUDENT"

ON "UNIVERSITY"."UNIV\_ID" = "STUDENT"."UNIV\_ID"

GROUP BY "UNIV\_NAME", "KURS"

2. Напишите запрос для получения списка преподавателей с указанием их учебных предметов.

SELECT "SURNAME","SUBJECT"."SUBJ\_ID", "SUBJ\_NAME"

FROM "LECTURER" JOIN "SUBJ\_LECT"

ON "LECTURER"."LECTURER\_ID" = "SUBJ\_LECT"."LECTURER\_ID"

JOIN "SUBJECT"

ON "SUBJECT"."SUBJ\_ID" = "SUBJ\_LECT"."SUBJ\_ID"

GROUP BY "SURNAME","SUBJECT"."SUBJ\_ID", "SUBJ\_NAME"

3. Напишите запрос для получения списка преподавателей с указанием нагрузки (суммарного количества часов) в каждом семестре.

SELECT "SURNAME", "HOUR", "SEMESTR"

FROM "LECTURER"

JOIN "SUBJ\_LECT"

ON "LECTURER"."LECTURER\_ID" = "SUBJ\_LECT"."LECTURER\_ID"

JOIN "SUBJECT"

ON "SUBJ\_LECT"."SUBJ\_ID" = "SUBJECT"."SUBJ\_ID"

GROUP BY "SURNAME", "HOUR", "SEMESTR"

4. Напишите запрос для получения списка университетов вместе с названиями преподаваемых в них предметов.

SELECT "UNIV\_NAME", "SUBJ\_NAME"

FROM "UNIVERSITY"

JOIN "LECTURER"

ON "UNIVERSITY"."UNIV\_ID" = "LECTURER"."UNIV\_ID"

JOIN "SUBJ\_LECT"

ON "LECTURER"."LECTURER\_ID" = "SUBJ\_LECT"."LECTURER\_ID"

JOIN "SUBJECT"

ON "SUBJ\_LECT"."SUBJ\_ID" = "SUBJECT"."SUBJ\_ID"

GROUP BY "UNIV\_NAME", "SUBJ\_NAME"

5. Напишите запрос для получения списка университетов с указанием суммарного количества аудиторных часов в каждом семестре.

SELECT "UNIV\_NAME", "HOUR", "SEMESTR"

FROM "UNIVERSITY"

JOIN "LECTURER"

ON "UNIVERSITY"."UNIV\_ID" = "LECTURER"."UNIV\_ID"

JOIN "SUBJ\_LECT"

ON "LECTURER"."LECTURER\_ID" = "SUBJ\_LECT"."LECTURER\_ID"

JOIN "SUBJECT"

ON "SUBJ\_LECT"."SUBJ\_ID" = "SUBJECT"."SUBJ\_ID"

GROUP BY "UNIV\_NAME", "HOUR", "SEMESTR"

6. Напишите запрос для получения списка университетов с указанием суммарного количества часов, отводимых на изучение каждого предмета.

SELECT "UNIV\_NAME", "HOUR", "SUBJ\_NAME"

FROM "UNIVERSITY"

JOIN "LECTURER"

ON "UNIVERSITY"."UNIV\_ID" = "LECTURER"."UNIV\_ID"

JOIN "SUBJ\_LECT"

ON "LECTURER"."LECTURER\_ID" = "SUBJ\_LECT"."LECTURER\_ID"

JOIN "SUBJECT"

ON "SUBJ\_LECT"."SUBJ\_ID" = "SUBJECT"."SUBJ\_ID"

GROUP BY "UNIV\_NAME", "HOUR", "SUBJ\_NAME"

7. Напишите запрос для получения списка преподавателей с указанием суммарного количества часов, отведенных для обучения каждому из предметов.

SELECT "SURNAME", "HOUR", "SUBJ\_NAME"

FROM "LECTURER"

JOIN "SUBJ\_LECT"

ON "LECTURER"."LECTURER\_ID" = "SUBJ\_LECT"."LECTURER\_ID"

JOIN "SUBJECT"

ON "SUBJ\_LECT"."SUBJ\_ID" = "SUBJECT"."SUBJ\_ID"

GROUP BY "SURNAME", "HOUR", "SUBJ\_NAME"

8. Напишите запрос для сортировки списка университетов по значениям максимальной стипендии, выплачиваемой студентам.

SELECT "UNIV\_NAME", MAX("STIPEND")

FROM "UNIVERSITY" JOIN "STUDENT"

ON "UNIVERSITY"."UNIV\_ID" = "STUDENT"."UNIV\_ID"

GROUP BY "UNIV\_NAME"

9. Напишите запрос для получения списка университетов вместе с фамилиями самых молодых студентов, обучающихся в них.

SELECT "UNIV\_NAME", MAX("BIRTHDAY")

FROM "UNIVERSITY" JOIN "STUDENT"

ON "UNIVERSITY"."UNIV\_ID" = "STUDENT"."UNIV\_ID"

GROUP BY "UNIV\_NAME"

10. Напишите запрос для получения списка университетов вместе с фамилиями студентов, получающих максимальную для каждого университета стипендию.

SELECT "UNIV\_NAME", "STIPEND", "SURNAME"

FROM "UNIVERSITY" JOIN "STUDENT"

ON "UNIVERSITY"."UNIV\_ID" = "STUDENT"."UNIV\_ID"

WHERE ("STUDENT"."UNIV\_ID", "STIPEND") IN

(SELECT "UNIV\_ID", MAX("STIPEND")

FROM "STUDENT"

GROUP BY "UNIV\_ID")

ORDER BY "UNIV\_NAME", "SURNAME"

**2.6 Создание, изменение добавление данных и таблиц**

**УПРАЖНЕНИЕ 1**

1. Напишите команду, которая вводит в таблицу SUBJECT строку для нового предмета обучения со следующими значениями полей: SEMESTER = 4; SUBJ\_NAME = 'Алгебра'; HOUR = 72; SUBJ\_ID = 201.

INSERT INTO "SUBJECT"("SEMESTR", "SUBJ\_NAME", "HOUR", "SUBJ\_ID")

VALUES ('4', 'Алгебра', '72', '201')

2. Введите запись для нового студента, которого зовут Орлов Николай, обучающегося на первом курсе ВГУ, живущего в Воронеже, сведения о дате рождения и размере стипендии неизвестны.

INSERT INTO "STUDENT"("SURNAME", "NAME", "KURS", "CITY", "UNIV\_ID")

VALUES ('Орлов', 'Николай', '1','Воронеж','10')

3. Напишите команду, удаляющую из таблицы EXAMMARKS записи обо всех оценках студента, идентификатор которого равен 100.

DELETE FROM "EXAM\_MARKS"

WHERE "STUDENT\_ID" = 100

4. Напишите команду, которая увеличивает на 5 значение рейтинга всех университетов, расположенных в Санкт-Петербурге.

UPDATE "UNIVERSITY"

SET "RATING" = "RATING"+5

WHERE "CITY" = 'Санкт-Петербург'

5. Измените в таблице значение города, в котором проживает студент Иванов, на 'Воронеж'.

UPDATE "STUDENT"

SET "CITY" = 'Воронеж'

WHERE "SURNAME" = 'Иванов'

**УПРАЖНЕНИЕ 2**

1. Пусть существует таблица с именем STUDENT1, определения столбцов которой полностью совпадают с определениями столбцов таблицы STUDENT. Вставьте в эту таблицу сведения о студентах, успешно сдавших экзамены более чем по пяти предметам обучения.

SELECT "STUDENT"."STUDENT\_ID", "STUDENT"."SURNAME", "STUDENT"."NAME", "STUDENT"."STIPEND", "STUDENT"."KURS", "STUDENT"."CITY", "STUDENT"."BIRTHDAY", "STUDENT"."UNIV\_ID"

FROM "STUDENT"

WHERE "STUDENT"."STUDENT\_ID" in (

SELECT "STUDENT\_ID"

FROM "EXAM\_MARKS"

WHERE "MARK" >=3

GROUP BY "STUDENT\_ID"

HAVING COUNT("MARK") > 1);

2. Напишите команду, удаляющую из таблицы SUBJECT1 сведения о предметах обучения, по которым студентами не получено ни одной оценки.

DELETE FROM "SUBJECT1"

WHERE "SUBJ\_ID" NOT IN

(SELECT DISTINCT("SUBJECT"."SUBJ\_ID")

FROM "EXAM\_MARKS", "SUBJECT"

WHERE "SUBJECT"."SUBJ\_ID" = "EXAM\_MARKS"."SUBJ\_ID" AND "MARK" IS NOT NULL

)

3. Напишите запрос, увеличивающий размер стипендии на 20% всем студентам, у которых общая сумма баллов превышает значение 50.

UPDATE "STUDENT"

SET "STIPEND" = ("STIPEND" + (0.2 \* "STIPEND"))

WHERE 50 <=

(SELECT SUM("MARK")

FROM "EXAM\_MARKS"

WHERE "EXAM\_MARKS"."STUDENT\_ID" = "STUDENT"."STUDENT\_ID" )

**УПРАЖНЕНИЕ 3**

1. Напишите команду CREATE TABLE для создания таблицы LECTURER1.

CREATE TABLE "LECTURER1"

("LECTURER\_ID" INTEGER,

"SURNAME" VARCHAR(60),

"NAME" VARCHAR (60),

"CITY" VARCHAR(60),

"UNIV\_ID" INTEGER);

2. Напишите команду CREATE TABLE для создания таблицы SUBJECT1

CREATE TABLE "SUBJ\_LECT2"

("SUBJ\_ID" INTEGER,

"SUBJ\_NAME" VARCHAR(60),

"KURS" INTEGER,

"SEMESTR" INTEGER);

3. Напишите команду CREATE TABLE для создания таблицы UNIVERSITY1.

CREATE TABLE "UNIVERSITY1"

("UNIV\_ID" INTEGER,

"UNIV\_NAME" VARCHAR(60),

"RATING" INTEGER,

"CITY" VARCHAR(60));

4. Напишите команду CREATE TABLE для создания таблицы EXAM\_MARKS1.

CREATE TABLE "EXAM\_MARKS1"

("EXAM\_ID" INTEGER,

"STUDENT\_ID" INTEGER,

"SUBJ\_ID" INTEGER,

"RATING" INTEGER,

"EXAM\_DATE" DATE);

5. Напишите команду CREATE TABLE для создания таблицы SUBJ\_LECT1.

CREATE TABLE "SUBJ\_LECT1"

("LECTURER\_ID" INTEGER,

"SUBJ\_ID" INTEGER);

**УПРАЖНЕНИЕ 4**

1. Создайте таблицу EXAM\_MARKS так, чтобы не допускался ввод в таблицу двух записей об оценках одного студента по конкретным экзамену и предмету обучения.

CREATE TABLE "EXAM\_MARKS\_NEW1"

("EXAM\_ID" INTEGER,

"STUDENT\_ID" INTEGER,

"SUBJ\_ID" INTEGER,

"MARK" INTEGER,

"EXAM\_DATE" DATE,

CONSTRAINT "EXAM\_OGR" PRIMARY KEY("STUDENT\_ID", "EXAM\_ID", "SUBJ\_ID"))

2. Создайте таблицу предметов обучения SUBJECT2 так, чтобы количество отводимых на предмет часов по умолчанию было равно 36, не допускались записи с отсутствующим количеством часов, поле SUBJ\_ID являлось первичным ключом таблицы, а значения семестров (поле SEMESTR) лежали в диапазоне от 1 до 12.

CREATE TABLE "SUBJECT2"

("SUBJ\_ID" INTEGER PRIMARY KEY,

"SUBJ\_NAME" VARCHAR,

"HOUR" INTEGER NOT NULL DEFAULT '36',

"SEMESTR" INTEGER CHECK("SEMESTR" IN (1,12)))

Лабораторная работа №4

Выборка данных с использованием операторов языка SQL

**Цель**: овладеть основными практическими навыками выборки данных из таблиц реляционных баз данных с использованием оператора SELECT языка SQL.

Ход работы:

1. БД “Авто”
2. Напишите запрос, который выбирает все автомобили определенного цвета и находит средний год их выпуска. Цвет задается в виде параметра (например, «белый»). Нулевые не учитывать.

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_average\_year\_by\_color(color\_param TEXT)  RETURNS NUMERIC  LANGUAGE plpgsql  AS  '  BEGIN  RETURN (SELECT AVG("AUTO"."YEAR\_A") AS average\_year  FROM "AUTO"  JOIN "MENU" ON "AUTO"."C\_COLOR" = ("MENU"."C\_MENU" || "MENU"."C\_REC")  WHERE "MENU"."NAME\_REC" = color\_param  AND "AUTO"."YEAR\_A" IS NOT NULL);  END;  ';  SELECT get\_average\_year\_by\_color('белый'); |

1. Напишите запрос, который выбирает все автомобили, произведенные на заданном заводе–изготовителя и находит количество номеров тех.паспорта начинающихся на символы «ШП». Завод-изготовителя задается в виде параметра (например, «АЗЛК, СССР»).

|  |
| --- |
| **CREATE OR REPLACE FUNCTION count\_passports\_by\_plant(plant\_param TEXT)**  **RETURNS INTEGER**  **LANGUAGE plpgsql AS**  **'**  **BEGIN**  **RETURN (SELECT COUNT("AUTO"."N\_PASS")**  **FROM "AUTO"**  **INNER JOIN "MENU" ON "AUTO"."C\_PLANT" = ("MENU"."C\_MENU" || "MENU"."C\_REC")**  **WHERE "MENU"."NAME\_REC" = plant\_param**  **AND "AUTO"."N\_PASS" LIKE ''ШП%'');**  **END;**  **';**  **SELECT count\_passports\_by\_plant('АЗЛК, СССР');** |

1. Напишите запрос, который выбирает все автомобили определенной марки (модели) и находит количество автомобилей, номер которых не заканчивается на буквы «дб». Нулевые не учитывать. Марка задается в виде параметра (например, «москвич М-412»).

|  |
| --- |
| CREATE OR REPLACE FUNCTION count\_autos\_by\_brand(brand\_param TEXT)  RETURNS INTEGER  LANGUAGE plpgsql AS '  BEGIN  RETURN (SELECT COUNT(\*) AS count\_of\_auto  FROM "AUTO"  JOIN "MENU" ON "AUTO"."C\_MARK" = ("MENU"."C\_MENU" || "MENU"."C\_REC")  WHERE "MENU"."NAME\_REC" = brand\_param  AND "AUTO"."SIGN" NOT LIKE ''%дб''  AND "AUTO"."SIGN" IS NOT NULL);  END;  ';  -- Вызов функции  SELECT count\_autos\_by\_brand('москвич М-412'); |

1. Напишите запрос, который выводит год, в котором было выпущено наибольшее количество автомобилей. Учитываются только года с 1991 по 1993 включительно (используя предикат Between).

|  |
| --- |
| SELECT "YEA\_"  FROM "AUTO"  WHERE "YEA\_" BETWEEN 1991 AND 1993  GROUP BY "YEA\_"  ORDER BY COUNT("INV") DESC  LIMIT 1 |

1. Напишите запрос, который выбирает все автомобили заданной штатной группы, а также все автомобили, для которых штатная группа не задана (использовать внешнее объединение) – для них вывести в поле строк «Штатная группа не задана». Штатная группа задается параметром (например, «стандартный»).

|  |
| --- |
| SELECT "AUTO"."INV", COALESCE("AUTO"."C\_GROUP", 'Штатная группа не задана') AS "Group"  FROM "AUTO"  LEFT JOIN "MENU" ON "AUTO"."C\_GROUP" = ("MENU"."C\_MENU" || "MENU"."C\_REC")  WHERE "MENU"."NAME\_REC" = 'стандартный'  OR "AUTO"."C\_GROUP" IS NULL; |

1. Напишите запрос, который выбирает все автомобили заданного класса и заданного цвета. Класс и цвет вводятся как параметры (например, «средний класс» и «белый»).

|  |
| --- |
| SELECT \*  FROM "AUTO" INNER JOIN "MENU" ON "AUTO"."C\_CLASS" = ("MENU"."C\_MENU" || "MENU"."C\_REC")  WHERE LOWER("MENU"."NAME\_REC") IN ('средний класс', 'белый') |

1. Напишите запрос, который считает среднюю стоимость для каждого из класса автомобилей.

|  |
| --- |
| SELECT  "C\_CLASS" AS class\_of\_auto,  ROUND(AVG("COST")::numeric, 2) AS cost  FROM "AUTO"  WHERE "C\_CLASS" IS NOT NULL  GROUP BY "C\_CLASS" |

1. Определить количество автомобилей заданного владельца, номер гос. регистрации которых начинается на «00» и не заканчивается на «ду».

|  |
| --- |
| SELECT  COUNT(\*) AS count\_of\_auto,  "C\_OWNER" AS owner  FROM "AUTO"  WHERE "SIGN" LIKE '00%ду'  GROUP BY "C\_OWNER" |

1. Напишите запрос, который выводит все автомобили определенной марки с заданным типом кузова. Марка и кузов задаются как параметр (например, «М-67-36 б/коляски» и «седан»).

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_id\_from\_menu\_by\_text(brand\_param TEXT)  RETURNS TEXT  LANGUAGE plpgsql AS  '  BEGIN  RETURN (SELECT "C\_MENU" || "C\_REC"  FROM "MENU"  WHERE "NAME\_REC" = brand\_param);  END;  ';  SELECT \*  FROM "AUTO"  WHERE "C\_MARK" = get\_id\_from\_menu\_by\_text('ВАЗ-21061')  OR "C\_BODY" = get\_id\_from\_menu\_by\_text('хечбек') |

1. Напишите запрос, который выводит название марки, средняя цена автомобиля для которой наибольшая. Нулевые не учитывать.

|  |
| --- |
| SELECT "C\_MARK"  FROM "AUTO"  GROUP BY "C\_MARK"  ORDER BY AVG("COST") DESC  LIMIT 1 |

1. Вывести автомобиль(и), у которых наибольший период от ввода в эксплуатацию до даты тех. паспорта среди автомобилей, номера которых заканчиваются на «рду».

|  |
| --- |
| SELECT \*  FROM "AUTO"  WHERE "SIGN" LIKE '%рду' AND "D\_PASS" - "D\_EXPL" = (  SELECT MAX("D\_PASS" - "D\_EXPL")  FROM "AUTO"  WHERE "SIGN" LIKE '%рду'  ); |

1. По каждой штатной группе автомобилей определить сколько а/м каждой марки было выпущено в заданном году. Вывести названия групп и названия марок на экран. Год задается параметром (например, 1989).

|  |
| --- |
| SELECT  "C\_GROUP" AS code\_of\_group,  "C\_MARK" AS code\_of\_mark,  COUNT("C\_MARK") AS count\_of\_marks  FROM "AUTO"  WHERE "YEAR\_A" = 1991  GROUP BY "C\_GROUP", "C\_MARK"; |

1. Определить какие а/м заданного класса получили тех. паспорт после указанной даты. Указать также инвентарный номер автомобиля. Задать класс и дату (например, «средний класс» и 10.01.1989).

|  |
| --- |
| SELECT \*  FROM "AUTO" INNER JOIN "MENU" ON "AUTO"."C\_CLASS" = "MENU"."C\_MENU" || "MENU"."C\_REC"  WHERE "AUTO"."D\_PASS" > '1989-01-10' AND "MENU"."NAME\_REC" = 'средний класс' |

1. Написать запрос, который выводит номер автомобиля и название его класса. Если класс не определен, то вывести сообщение «класс не определен» (использовать left join).

|  |
| --- |
| SELECT "AUTO"."INV", COALESCE("MENU"."NAME\_REC", 'класс не определён')  FROM "AUTO"  LEFT JOIN "MENU" ON "AUTO"."C\_CLASS" = ("MENU"."C\_MENU" || "MENU"."C\_REC") |

1. Определить, сколько автомобилей каждой марки имеют год выпуска меньший, чем округленный до целого средний год выпуска автомобилей заданной пользователем марки (использовать подзапрос). Задать название марки (например, «М-67-36 б/коляски»).

|  |
| --- |
| SELECT  "AUTO"."C\_MARK", COUNT("AUTO"."C\_MARK")  FROM "AUTO"  WHERE "AUTO"."YEA\_" < (SELECT ROUND(AVG("AUTO"."YEA\_"))  FROM "AUTO"  LEFT JOIN "MENU" ON "AUTO"."C\_CLASS" = ("MENU"."C\_MENU" || "MENU"."C\_REC")  WHERE "MENU"."NAME\_REC" = 'особо малый класс')  GROUP BY "AUTO"."C\_MARK" |

1. Определить какое количество автомобилей каждой марки в каком году было произведено (использовать перекрестный запрос). Нулевые марки не учитывать.

|  |
| --- |
| SELECT  a1."C\_MARK" AS mark,  COUNT(a1."C\_MARK") AS count\_of\_auto,  a2."YEA\_" AS year  FROM "AUTO" AS a1 CROSS JOIN "AUTO" AS a2  GROUP BY a1."C\_MARK", a2."YEA\_" |

1. Определить какое количество автомобилей каждого цвета к какой штатной группе относится (использовать перекрестный запрос). Нулевые штатные группы и цвета не учитывать.

|  |
| --- |
| SELECT  a1."C\_GROUP" AS group\_of\_auto,  "MENU"."NAME\_REC" AS name\_of\_color,  COUNT("MENU"."NAME\_REC") AS count\_of\_auto  FROM "AUTO" AS a1 CROSS JOIN "AUTO" AS a2 INNER JOIN "MENU" ON a2."C\_COLOR" = ("MENU"."C\_MENU" || "MENU"."C\_REC")  WHERE a1."C\_GROUP" IS NOT NULL  GROUP BY a1."C\_GROUP", "MENU"."NAME\_REC" |

1. Вывести сумму, которую потратил каждый владелец на автомобили. Нулевые цены не учитывать.

|  |
| --- |
| SELECT  "AUTO"."C\_OWNER" AS owner,  SUM("AUTO"."COST") AS summary\_on\_auto  FROM "AUTO"  WHERE "AUTO"."COST" IS NOT NULL AND "AUTO"."C\_OWNER" IS NOT NULL  GROUP BY "AUTO"."C\_OWNER" |

1. Вывести все автомобили, в номерах которых есть сочетание «3р», а номер тех. паспорта начинается на «ВМV».

|  |
| --- |
| SELECT "INV","N\_PASS", "SIGN"  FROM "AUTO"  WHERE "N\_PASS" LIKE 'BMV%' AND "SIGN" LIKE '%3р%' |

1. Вывести количество владельцев автомобилей, заданного цвета. Цвет вводится с клавиатуры.

|  |
| --- |
| SELECT  "AUTO"."C\_OWNER" AS owners,  COUNT(\*) AS count\_of\_auto  FROM "MENU" INNER JOIN "AUTO" ON "AUTO"."C\_COLOR" = "MENU"."C\_MENU" || "MENU"."C\_REC"  WHERE "MENU"."NAME\_REC" = 'зеленый'  GROUP BY "AUTO"."C\_OWNER" |

Б) БД “Деканат”

1. Выведите пронумерованный список ста первых студентов, у которых средний итоговый балл (StudPlany.ResultMark) за весь период обучения наибольший (без учета пустых значений). Список отсортировать по алфавиту. Вывести: Фамилию И.О., зачетная книжка, средний итоговый балл, позиция в списке 100.

|  |
| --- |
| WITH AverageMarksOfStudents AS (  SELECT  "StudentsT"."IDStudent" AS student\_id,  AVG(CAST("StudPlany"."ResultMark" AS NUMERIC)) AS average\_mark\_of\_student  FROM "StudentsT" INNER JOIN "StudPlany" ON "StudentsT"."IDStudent" = "StudPlany"."Student"  GROUP BY "StudentsT"."IDStudent"  ),  RankedStudents AS (  SELECT  "Persons"."Фам",  "Persons"."Имя",  "Persons"."Отч",  "PersonT"."Номер зачетки",  ROW\_NUMBER() OVER (ORDER BY "Persons"."Фам", "Persons"."Имя", "Persons"."Отч") AS position  FROM "Persons"  INNER JOIN AverageMarksOfStudents ON "Persons"."IDPersons" = AverageMarksOfStudents.student\_id  INNER JOIN "PersonT" ON "Persons"."IDPersons" = "PersonT"."IDPerson"  )  SELECT  RankedStudents."Фам",  RankedStudents."Имя",  RankedStudents."Отч",  RankedStudents."Номер зачетки",  position  FROM RankedStudents  WHERE position <= 100  ORDER BY RankedStudents."Фам", RankedStudents."Имя", RankedStudents."Отч"; |

1. Для обучающихся студентов посчитать количество дисциплин (без повторов), которые будут им преподаваться в следующем семестре (текущий семестр установлен у объекта «Студент»).

|  |
| --- |
| SELECT  "StudentsT"."IDStudent" AS student\_id,  COUNT(DISTINCT "Plany"."PDiscipline") AS count\_of\_subjects  FROM "StudentsT"  INNER JOIN "StudPlany" ON "StudentsT"."IDStudent" = "StudPlany"."Student"  INNER JOIN "Plany" ON "StudPlany"."GroopPlan" = "Plany"."IDPlany"  WHERE "StudentsT"."Semestr" + 1 = "Plany"."PSemestr"  GROUP BY "StudentsT"."IDStudent"; |

1. По каждому человеку вывести количество всех зарегистрированных приказов (т.е. дата регистрации не пустая), а также количеств зарегистрированных приказов, которые были проведены.

|  |
| --- |
| SELECT "Persons"."IDPersons",  COUNT(DISTINCT "Приказы"."Код Приказа") AS registered\_orders,  COUNT(CASE WHEN "Приказы"."Проведен" = 'true' THEN 1 END) AS registered\_orders\_that\_have\_been\_carried\_out  FROM "Persons"  JOIN "PersonT"  ON "Persons"."IDPersons" = "PersonT"."IngPerson"  JOIN "StudentsT"  ON "PersonT"."IDPerson" = "StudentsT"."IDPerson"  JOIN "ПриказыПоСтудентам"  ON "StudentsT"."IDStudent" = "ПриказыПоСтудентам"."КодСтудента"  JOIN "Приказы"  ON "ПриказыПоСтудентам"."КодПриказа" = "Приказы"."Код Приказа"  WHERE "Приказы"."ДатаРегистрации" IS NOT NULL  GROUP BY "Persons"."IDPersons" |

1. Для всех записей о странах в таблице Addresses вывести количество человек (если нет, то 0), которые имеют эту страну в качестве своего адреса. Поиск проводить для всех мальчиков 4го курса и для девушек 9го семестра (только обучающиеся студенты).

|  |
| --- |
| SELECT  "Addresses"."AddrCountry" AS country,  COUNT("Persons"."IDPersons") AS count\_of\_students  FROM "Persons"  INNER JOIN "Addresses" ON "Persons"."IDPersons" = "Addresses"."IDВладельца"  INNER JOIN "StudentsT" ON "Persons"."IDPersons" = "StudentsT"."IDPerson"  WHERE  (("Persons"."Пол" = 'м' AND "StudentsT"."Semestr" IN (7, 8))  OR  ("Persons"."Пол" = 'ж' AND "StudentsT"."Semestr" = 9))  GROUP BY "Addresses"."AddrCountry" |

1. Вывести преподавателей и дисциплины, по которым в таблице оценок для обучающихся студентов в текущем семестре есть незаполненные (пустые) оценки.

|  |
| --- |
| SELECT  "Prepod"."PrepodFamIO" AS teacher\_name,  "Plany"."PDiscipline" AS discipline\_code  FROM  "Mark"  FULL JOIN "StudPlany" ON "Mark"."TStudPlan" = "StudPlany"."IDStudPlan"  FULL JOIN "Plany" ON "StudPlany"."GroopPlan" = "Plany"."IDPlany"  FULL JOIN "Prepod" ON "Plany"."IDPlany" = "Prepod"."PlanyID"  FULL JOIN "StudentsT" ON "StudPlany"."Student" = "StudentsT"."IDStudent"  WHERE  "Mark"."Mark" IS NULL AND "StudentsT"."Semestr" = "Plany"."PSemestr"; |

1. По каждому студенту вывести отношение количества пустых оценок к общему количеству оценок для этого студента. Формат вывода: номер з/к, % пустых.

|  |
| --- |
| SELECT  pt."Номер зачетки",  ROUND(  COUNT(CASE WHEN sp."ResultMark" IS NULL THEN 1 END) \* 100.0 /  NULLIF(COUNT(sp."ResultMark"), 0), 2  ) AS "% пустых"  FROM  "PersonT" pt  JOIN  "StudentsT" st ON pt."IDPerson" = st."IDPerson"  LEFT JOIN  "StudPlany" sp ON st."IDStudent" = sp."Student"  GROUP BY  pt."Номер зачетки"; |

1. Вывести первых 5 преподавателей, сумма аудиторных часов за текущий семестр у которых наибольшая. Список пронумеровать в порядке убывания количества аудиторных часов.

|  |
| --- |
| SELECT "IDPrepod", "PrepodFamIO", SUM("Plany"."PAuditTime")  FROM "Prepod"  JOIN "Plany"  ON "Prepod"."IDPrepod" = "Plany"."IDPlany"  WHERE "Plany"."PSemestr" = 4  GROUP BY "Prepod"."IDPrepod"  ORDER BY SUM("Plany"."PAuditTime") DESC  LIMIT 5 |

1. Найти всех отчисленных студентов, у которых есть хотя бы одна задолженность.

|  |
| --- |
| SELECT  "StudentsT"."IDStudent" AS student\_id,  "Persons"."Фам" || ' ' || "Persons"."Имя" || ' ' || "Persons"."Отч" AS student\_name,  "StudPlany"."ResultMark" AS result\_mark  FROM  "StudentsT"  JOIN "PersonT" ON "StudentsT"."IDPerson" = "PersonT"."IDPerson"  JOIN "Persons" ON "PersonT"."IDPerson" = "Persons"."IDPersons"  JOIN "StudPlany" ON "StudentsT"."IDStudent" = "StudPlany"."Student"  WHERE  -- "StudentsT"."LearnStatus" = 'D' -- статус отчисленных студентов (предположительно "D")  -- AND ("StudPlany"."ResultMark" IS NULL OR "StudPlany"."ResultMark" < 50);  "StudPlany"."ResultMark" IS NULL |

1. Найти всех братьев и сестер и вывести разницу в годах, которую посчитать как дата рождения самого младшего за вычетом даты рождения самого старшего. За признак родства принимать фамилию и отчество.

|  |
| --- |
| WITH NormalizedPatronymics AS (  SELECT  "Фам",  CASE  WHEN RIGHT("Отч", 1) = 'а' THEN LEFT("Отч", LENGTH("Отч") - 1) -- Убираем последнюю букву 'а' для женских отчеств  ELSE "Отч" -- Оставляем мужские отчества без изменений  END AS NormalizedОтч,  "ДатаРожд"  FROM  "Persons"  ),  Siblings AS (  SELECT  "Фам",  "normalizedОтч",  MIN("ДатаРожд") AS MinBirthDate,  MAX("ДатаРожд") AS MaxBirthDate,  COUNT(\*) AS SiblingCount  FROM  NormalizedPatronymics  GROUP BY  "Фам", "normalizedОтч"  HAVING  COUNT(\*) > 1 -- Учитываем только тех, у кого есть братья/сестры  )  SELECT  "Фам",  "normalizedОтч" AS "Отч",  EXTRACT(YEAR FROM AGE(MaxBirthDate, MinBirthDate)) AS AgeDifference  FROM  Siblings; |

1. Найти всех студентов выпускного курса (10-й семестр), которые претендуют на получение красного диплома, если известно, что для получения красного диплома у студента должно быть не более 15% отметок «хорошо» по экзаменам и диф. зачетам (остальные отметки «отлично»).

|  |
| --- |
| WITH StudentMarks AS (  SELECT  st."IDStudent",  st."IDPerson",  COUNT(\*) AS TotalMarks,  COUNT(CASE WHEN m."Mark" = 4 THEN 1 END) AS GoodMarks,  COUNT(CASE WHEN m."Mark" = 5 THEN 1 END) AS ExcellentMarks  FROM  "StudentsT" st  JOIN  "StudPlany" sp ON st."IDStudent" = sp."Student"  JOIN  "Mark" m ON sp."IDStudPlan" = m."TStudPlan"  WHERE  st."Semestr" = 10 -- Выпускной курс (10-й семестр)  AND (m."ExamType" = 'экзамен' OR m."ExamType" = 'диф. зачет') -- Учитываем только экзамены и диф. зачеты  GROUP BY  st."IDStudent", st."IDPerson"  )  SELECT  sm."IDStudent",  p."Фам",  p."Имя",  p."Отч",  sm.TotalMarks,  sm.GoodMarks,  sm.ExcellentMarks,  ROUND((sm.GoodMarks \* 100.0 / sm.TotalMarks), 2) AS GoodMarksPercentage  FROM  StudentMarks sm  JOIN  "Persons" p ON sm."IDPerson" = p."IDPersons"  WHERE  sm.GoodMarks \* 100.0 / sm.TotalMarks <= 15; -- Условие для красного диплома |

Лабораторная работа № 5

Физическая реализация разрабатываемой базы данных, манипуляция и изменение данных с использованием операторов языка SQL

**Цель**: овладеть основными практическими навыками манипуляции данными в таблицах реляционных баз данных с использованием операторов языка SQL.

Ход работы:

1. Реализовать скрипт создания разрабатываемой базы данных с использованием инструкций части DDL языка SQL (CREATE TABLE, CREATE INDEX и т.д.);
2. При создании таблиц не забывать указывать необходимые ограничения целостности и значения по умолчанию (UNIQUE, DEFAULT, CHECK, PRIMARY KEY, FOREIGN KEY и т.д.);

|  |
| --- |
| -- Удаление существующих таблиц --  DROP TABLE IF EXISTS books CASCADE;  DROP TABLE IF EXISTS readers CASCADE;  DROP TABLE IF EXISTS libraries CASCADE;  DROP TABLE IF EXISTS employees CASCADE;  DROP TABLE IF EXISTS book\_issue CASCADE;  CREATE TABLE books (  book\_id INT PRIMARY KEY,  title VARCHAR(255),  author VARCHAR(255),  isbn VARCHAR(20),  publication\_date DATE,  status VARCHAR(50)  );  CREATE TABLE readers (  reader\_id INT PRIMARY KEY,  surname VARCHAR(100),  name VARCHAR(100),  patronymic VARCHAR(100),  contact\_info TEXT,  registration\_date DATE  );  CREATE TABLE libraries (  library\_id INT PRIMARY KEY,  name VARCHAR(200),  address VARCHAR(300),  phone VARCHAR(20)  );  CREATE TABLE employees (  employee\_id INT PRIMARY KEY,  surname VARCHAR(100),  name VARCHAR(100),  patronymic VARCHAR(100),  position VARCHAR(100),  library\_id INT,  FOREIGN KEY (library\_id) REFERENCES libraries(library\_id)  );  CREATE TABLE book\_issue (  issue\_id INT PRIMARY KEY,  reader\_id INT,  employee\_id INT,  library\_id INT,  book\_id INT,  issue\_date DATE,  return\_date DATE,  status VARCHAR(50),  FOREIGN KEY (reader\_id) REFERENCES readers(reader\_id),  FOREIGN KEY (employee\_id) REFERENCES employees(employee\_id),  FOREIGN KEY (library\_id) REFERENCES libraries(library\_id),  FOREIGN KEY (book\_id) REFERENCES books(book\_id)  ); |

1. Написать запросы добавления, изменения и удаления данных из БД, оформить их в виде функций;

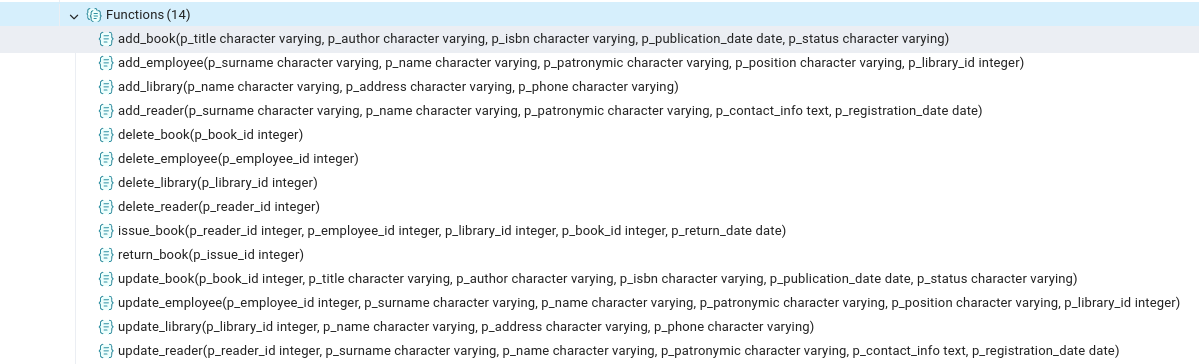


Рисунок 5.1 – Функции

Лабораторная работа №6

Разработка клиентского интерфейса для работы с базой данных

**Цель**: научиться разрабатывать интерфейс конечного пользователя для реляционных БД с использованием стандартных программных интерфейсов.

Снизу скришнот схемы БД, спроектированной в первой лабораторной работе (рисунок 6.1)

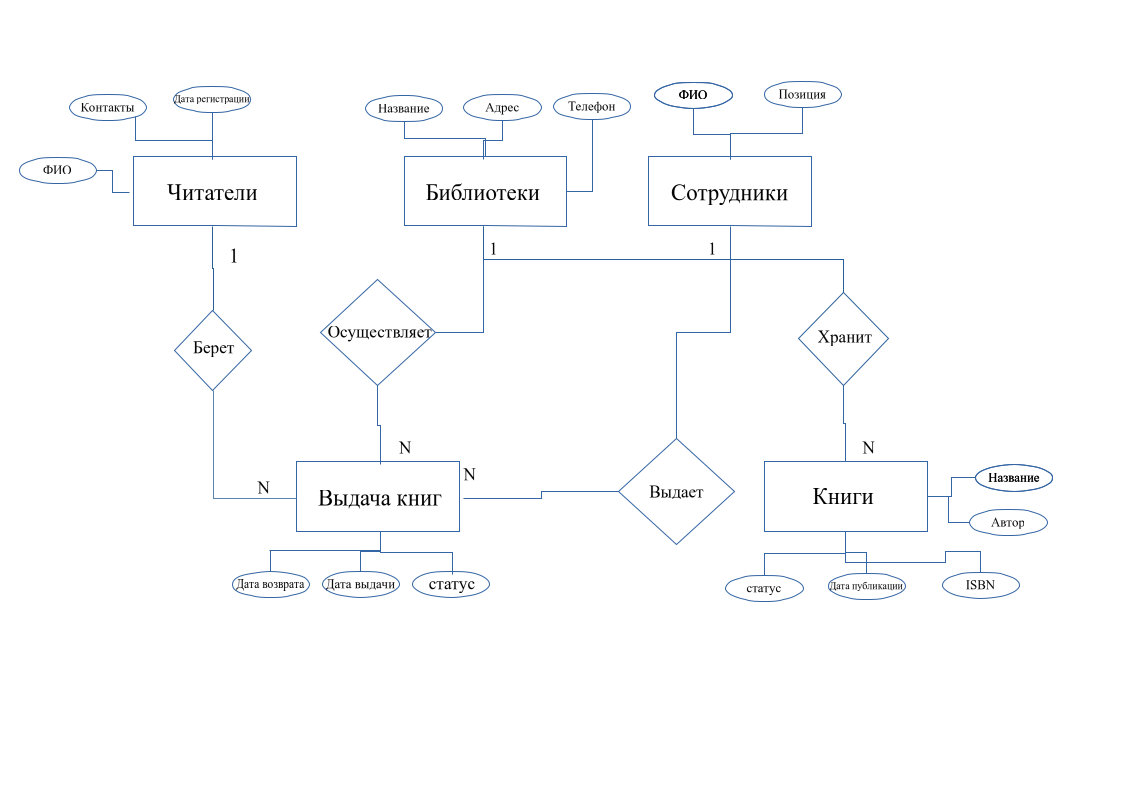
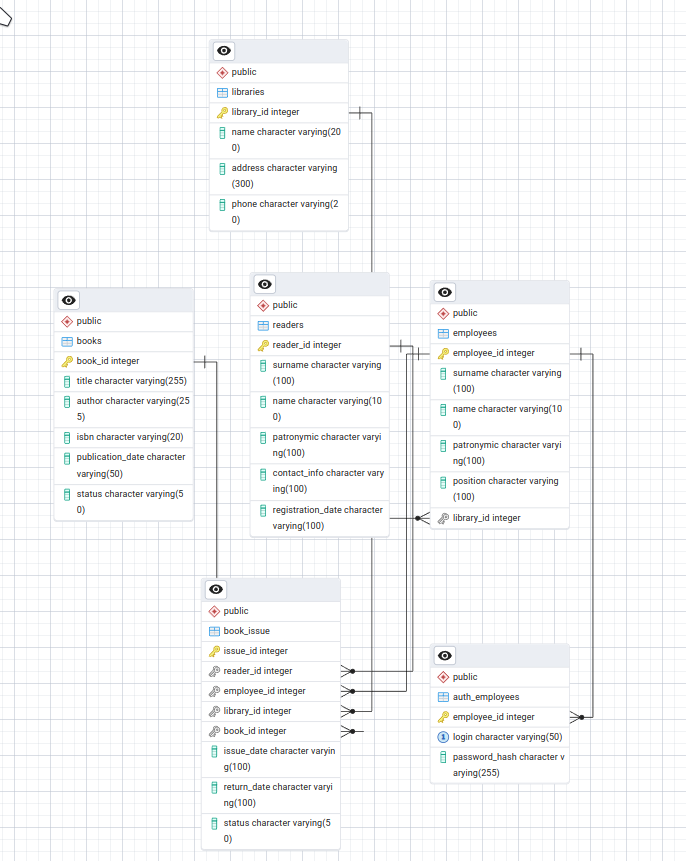


Рисунок 6.1

На рисунке 6.2 представлена UML-диаграмма



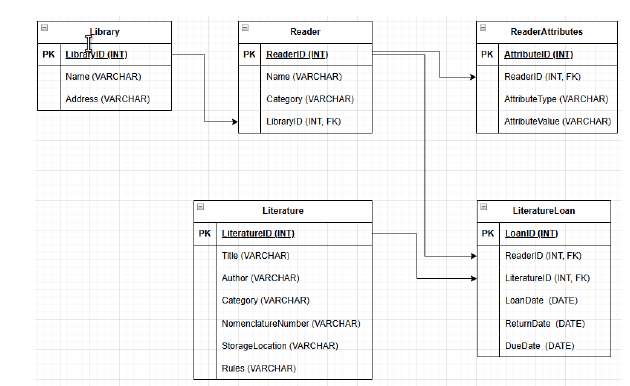


Рисунок 6.2

На рисунках 6.3-6.5 показана базовая работа с БД

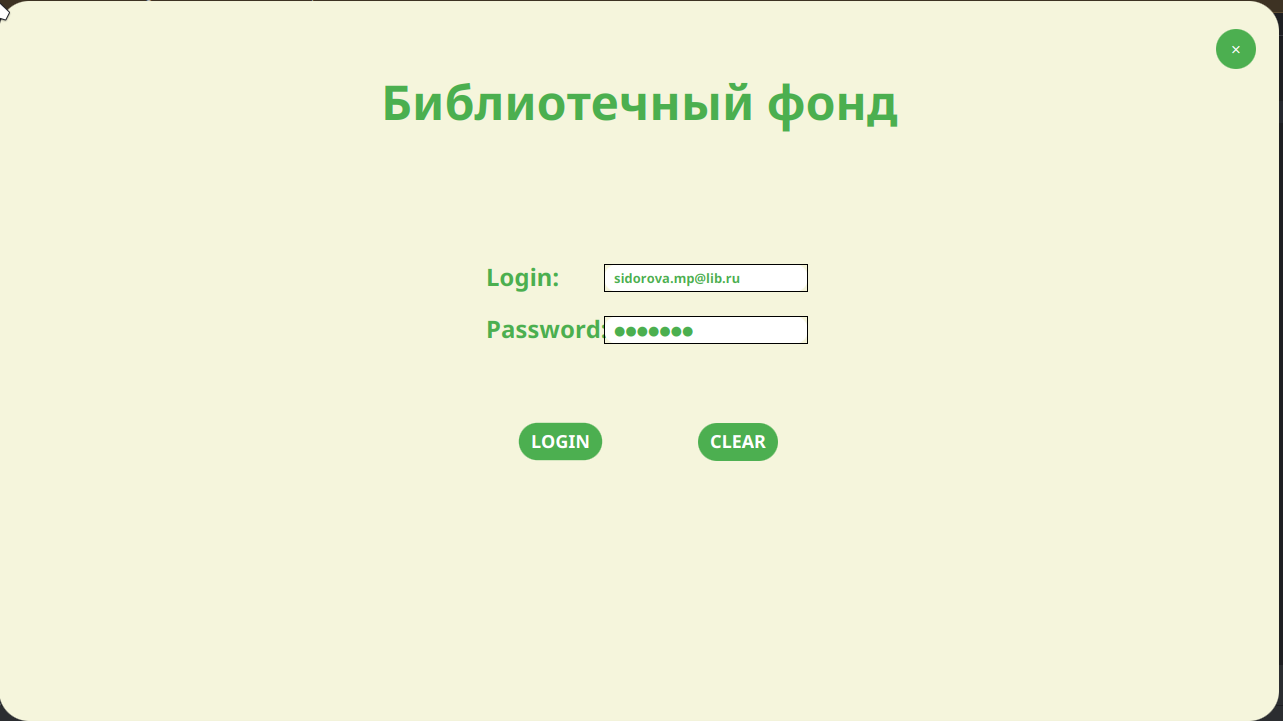


Рисунок 6.3

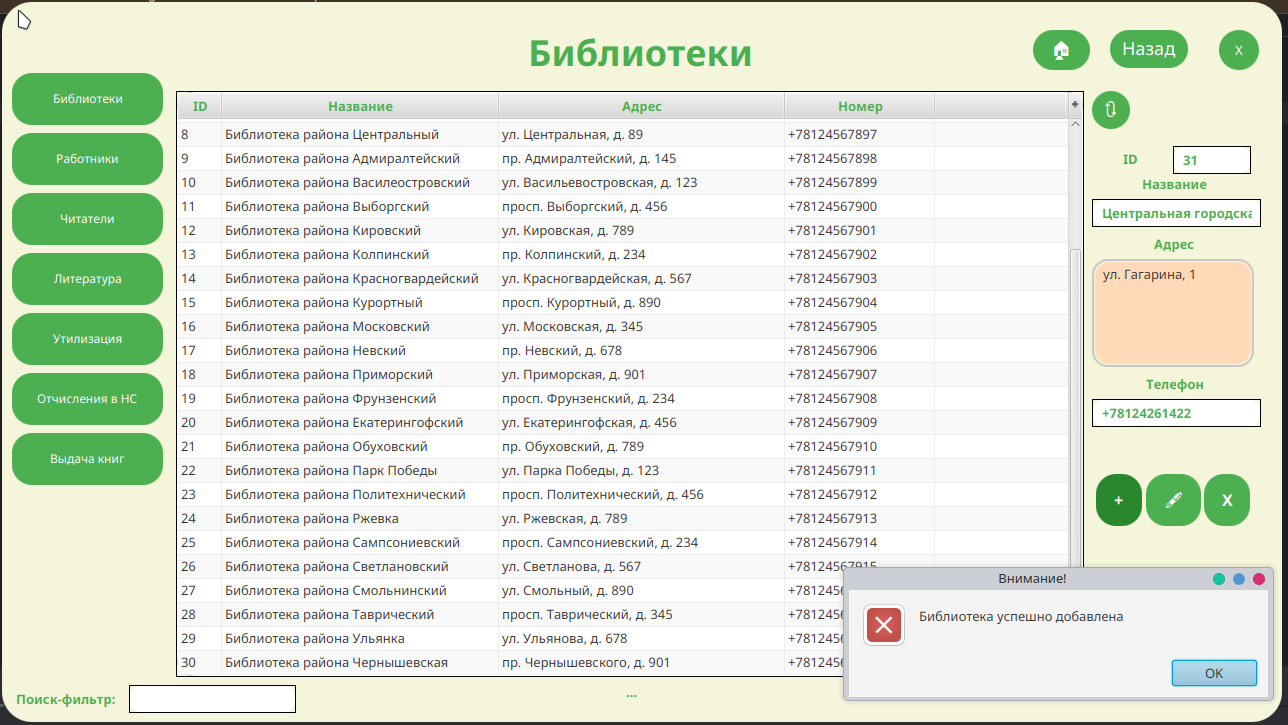


Рисунок 6.4

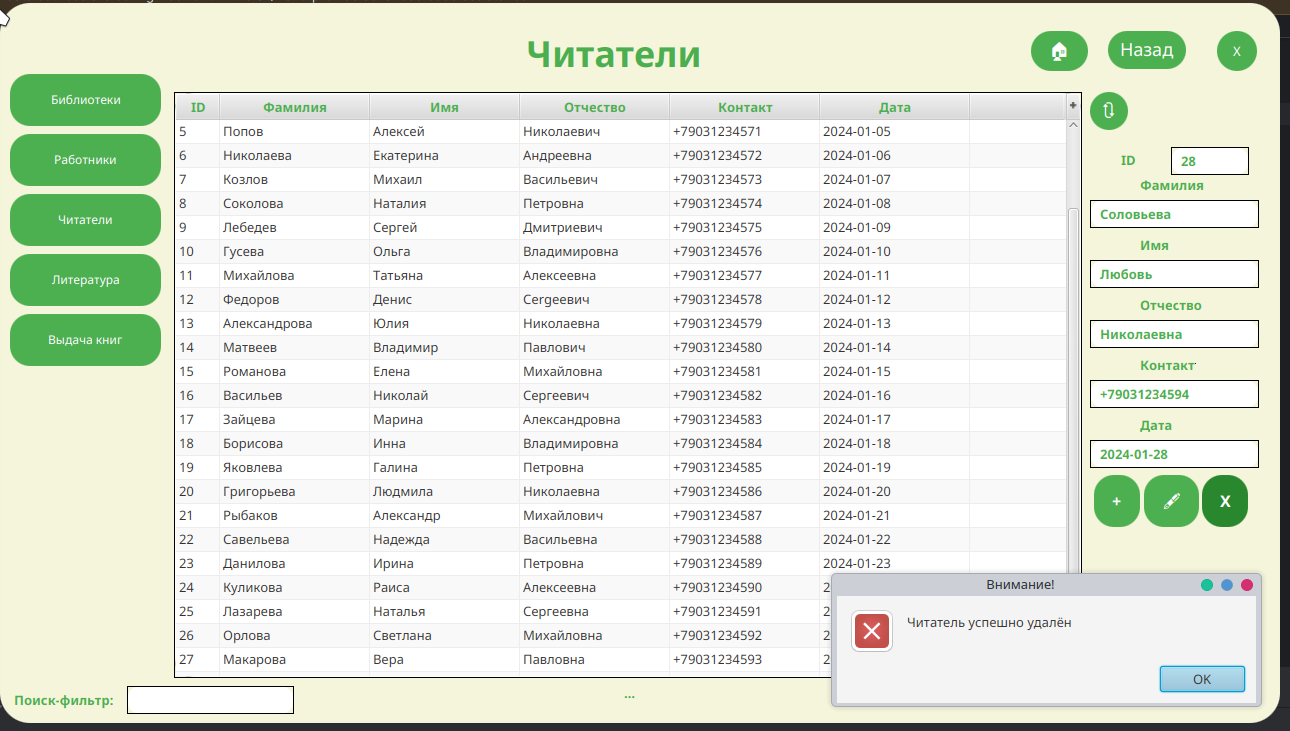


Рисунок 6.5

Контрольные вопросы:

**1. Что такое транзакция?**

**Транзакция** — это последовательность операций над базой данных, которая рассматривается как единое логическое целое. Транзакция должна удовлетворять свойствам **ACID**:

* **Атомарность (Atomicity)** — либо все операции транзакции выполняются, либо ни одна.
* **Согласованность (Consistency)** — транзакция переводит БД из одного согласованного состояния в другое.
* **Изолированность (Isolation)** — параллельные транзакции не влияют друг на друга.
* **Долговечность (Durability)** — после завершения транзакции её результаты сохраняются даже при сбое.

Пример: перевод денег между счетами (списание с одного и зачисление на другой должно быть одной транзакцией).

**2. Что такое журнализация?**

**Журнализация (logging)** — это механизм записи в специальный журнал (лог) всех изменений данных и служебной информации для восстановления БД после сбоев.

Основные цели:

* Восстановление данных после откатов транзакций.
* Обеспечение отказоустойчивости (например, при аварийном отключении сервера).

Примеры журналов:

* **Журнал транзакций** (запись операций до их выполнения — **WAL**).
* **Журнал отката (undo log)** — для отмены незавершённых транзакций.
* **Журнал повтора (redo log)** — для повторного выполнения завершённых транзакций.

**3. Что понимается под понятием «целостность БД»?**

**Целостность базы данных** — это корректность и непротиворечивость данных в БД, обеспечиваемая выполнением определённых правил и ограничений.

Виды целостности:

* **Физическая** — защита от аппаратных сбоев (через резервное копирование, журнализацию).
* **Логическая** — соблюдение бизнес-правил (например, возраст человека не может быть отрицательным).

Механизмы обеспечения:

* **Ограничения (constraints)** — PRIMARY KEY, FOREIGN KEY, CHECK, NOT NULL.
* **Триггеры** — автоматические действия при изменении данных.
* **Транзакции** — гарантия согласованного состояния после выполнения операций.

**4. Какие бывают виды сбоев? Охарактеризуйте их.**

1. **Аппаратные сбои**
   * **Отказ диска** (потеря данных).
   * **Сбой питания** (неожиданное отключение сервера).
   * **Ошибки оперативной памяти** (искажение данных).  
     *Решение:* резервное копирование, RAID-массивы, журнализация.
2. **Программные сбои**
   * **Ошибки в СУБД** (баги, приводящие к краху).
   * **Сбои в приложении** (некорректные запросы).  
     *Решение:* откат транзакций, использование WAL.
3. **Ошибки пользователей**
   * **Некорректные данные** (например, ввод отрицательной цены).
   * **Случайное удаление данных**.  
     *Решение:* ограничения (CHECK), транзакции, бэкапы.
4. **Сетевые сбои**
   * **Разрыв соединения** во время транзакции.  
     *Решение:* таймауты, механизмы завершения транзакций.

**5. Что такое протокол WAL?**

**WAL (Write-Ahead Logging)** — это протокол журнализации, при котором любые изменения данных **сначала записываются в журнал (лог)**, и только потом применяются к самой БД.

**Принцип работы:**

1. Перед изменением страницы данных в БД записывается запись в журнал (redo log).
2. Изменения применяются к данным только после подтверждения записи в лог.
3. При сбое СУБД может восстановить данные, повторно применив записи из журнала.

**Преимущества:**

* Гарантия **долговечности (durability)** данных.
* Ускорение работы (запись в лог обычно быстрее, чем прямое обновление БД).
* Возможность **восстановления после сбоев** (по журналу redo).

**Пример использования:**

* PostgreSQL, SQLite, Oracle и другие СУБД используют WAL для обеспечения отказоустойчивости.

Лабораторная работа №7

Изучение объектно-реляционных возможностей СУБД на примере PostgreSQL

**Цель**: научиться работать с расширенными возможностями реляционных СУБД на примере PostgreSQL.

Ход работы:

Требуется выполнить следующее:

1. Реализовать базовое отношение my\_object, содержащее атрибуты:

* id – идентификатор записи, тип: integer;
* time\_create – время создания записи (время рождения), тип timestamp;
* time\_dead – время смерти записи, тип timestamp.

Первичный ключ (Primary Key): {id, time\_create}

|  |
| --- |
| /\*\*  Реализовать базовое отношение my\_object, содержащее атрибуты:  • id – идентификатор записи, тип: integer;  • time\_create – время создания записи (время рождения), тип timestamp;  • time\_dead – время смерти записи, тип timestamp.  Первичный ключ (Primary Key): {id, time\_create}  \*/  DROP TABLE IF EXISTS my\_object CASCADE;  CREATE TABLE my\_object  (  id INT,  time\_create TIMESTAMP,  time\_dead TIMESTAMP,  CONSTRAINT prim PRIMARY KEY (id, time\_create)  ); |

2. Создать таблицы БД (от 3х до 5ти), объявив их наследниками my\_object;

|  |
| --- |
| /\*\*  Создать таблицы БД (от 3х до 5ти), объявив их наследниками my\_object;  \*/  DROP TABLE IF EXISTS electronic\_equipment CASCADE;  DROP TABLE IF EXISTS furniture CASCADE;  DROP TABLE IF EXISTS consumables CASCADE;  -- Таблица для техники. Техника может включать компьютеры, мониторы, принтеры и т.д.  CREATE TABLE electronic\_equipment (  warranty\_period INTEGER NOT NULL,  manufacturer VARCHAR(100)  ) INHERITS (my\_object);  -- Таблица для мебели. Мебель, такая как стулья, столы, шкафы.  CREATE TABLE furniture (  material VARCHAR(50) NOT NULL,  dimensions VARCHAR(50)  ) INHERITS (my\_object);  -- Таблица для расходных материалов. Расходные материалы, такие как канцелярия, бумага или картриджи.  CREATE TABLE consumables (  quantity INTEGER NOT NULL, -- Количество  unit VARCHAR(20) NOT NULL -- Единица измерения (шт., пачки и т.д.)  ) INHERITS (my\_object); |

3. Определить триггерную функцию(-и), которая(-ые) при добавлении новой записи в таблицу БД проверяет наличие такой записи среди существующих объектов (сравнение по id).

Если запись с таким id найдена, то для этой записи устанавливается время смерти, а добавляемая запись содержит в себе соответственные значения атрибутов найденной записи (кроме тех, которые заданы для новой записи), пустое значение времени смерти и новое значение времени рождения, равное текущему моменту. Таким образом, у новой записи время рождения равно, времени смерти ее предка.

Если запись с таким id на найдена, то происходит ее добавление в БД с установлением в качестве времени рождения текущего момента времени.

|  |
| --- |
| /\*\*  Определить триггерную функцию(-и), которая(-ые) при добавлении новой записи в таблицу БД проверяет наличие такой записи  среди существующих объектов (сравнение по id).  Если запись с таким id найдена, то для этой записи устанавливается время смерти, а добавляемая запись содержит в себе  соответственные значения атрибутов найденной записи (кроме тех, которые заданы для новой записи),  пустое значение времени смерти и новое значение времени рождения, равное текущему моменту.  Таким образом, у новой записи время рождения равно, времени смерти ее предка.  Если запись с таким id на найдена, то происходит ее добавление в БД с установлением в качестве времени рождения текущего  момента времени.  \*/  CREATE OR REPLACE FUNCTION handle\_insert\_object()  RETURNS TRIGGER AS $$  DECLARE  existing\_record my\_object%ROWTYPE;  BEGIN  -- Проверка наличия записи с таким же id и без времени смерти  SELECT \* INTO existing\_record  FROM my\_object  WHERE id = NEW.id AND time\_dead IS NULL  ORDER BY time\_create DESC  LIMIT 1;  IF FOUND THEN  -- Если запись найдена, установить время смерти предка  UPDATE my\_object  SET time\_dead = now()  WHERE id = existing\_record.id AND time\_create = existing\_record.time\_create;  -- Перенос значений времени и сброс времени смерти  NEW.time\_create = now(); -- Новое время создания записи  NEW.time\_dead = NULL; -- Сбрасываем время смерти  ELSE  -- Если запись не найдена, просто устанавливаем время создания  NEW.time\_create = now();  NEW.time\_dead = NULL;  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE OR REPLACE TRIGGER handle\_insert\_electronic\_equipment  BEFORE INSERT ON electronic\_equipment  FOR EACH ROW EXECUTE FUNCTION handle\_insert\_object();  CREATE OR REPLACE TRIGGER handle\_insert\_furniture  BEFORE INSERT ON furniture  FOR EACH ROW EXECUTE FUNCTION handle\_insert\_object();  CREATE OR REPLACE TRIGGER handle\_insert\_consumables  BEFORE INSERT ON consumables  FOR EACH ROW EXECUTE FUNCTION handle\_insert\_object();  -- Вставляем новую запись в таблицу furniture  INSERT INTO furniture (id, material, dimensions)  VALUES (1, 'Wood', '100x50x75');  -- Вставляем запись с таким же id (это вызовет обновление времени смерти первой записи)  INSERT INTO furniture (id, material, dimensions)  VALUES (1, 'Metal', '120x60x75');  -- Проверяем записи в my\_object  SELECT \* FROM my\_object;  -- Проверяем записи в furniture  SELECT \* FROM furniture; |

4. Напишите собственную агрегатную функцию, которая производит конкатенацию всех строк в атрибуте, передаваемом в функцию в качестве аргумента, с заданным в качестве параметра функции символом-разделителем. Например, если в качестве значений атрибутов-аргументов агрегатной функции будут выступать строки ‘123’, ‘456’ и ‘789’, а в качестве символа разделителя будет задан ‘\_’, то агрегатная функция вернет значение ‘123\_456\_789’ в результате группирования этих трех кортежей (порядок сортировки по возрастанию);

|  |
| --- |
| /\*\*  Напишите собственную агрегатную функцию, которая производит конкатенацию всех строк в атрибуте, передаваемом в функцию  в качестве аргумента, с заданным в качестве параметра функции символом-разделителем. Например, если в качестве значений  атрибутов-аргументов агрегатной функции будут выступать строки ‘123’, ‘456’ и ‘789’, а в качестве символа разделителя  будет задан ‘\_’, то агрегатная функция вернет значение ‘123\_456\_789’ в результате группирования этих трех кортежей  (порядок сортировки по возрастанию);  \*/  -- Функция для обновления состояния (конкатенация строк с разделителем)  CREATE OR REPLACE FUNCTION concat\_with\_delimiter\_agg(  state TEXT, -- Текущее состояние (аккумулятор)  value TEXT, -- Значение из текущей строки  delimiter TEXT -- Разделитель  )  RETURNS TEXT  LANGUAGE plpgsql AS  '  BEGIN  -- Если текущее состояние NULL, возвращаем текущее значение  IF state IS NULL THEN  RETURN value;  END IF;  -- Если текущее значение NULL, возвращаем состояние  IF value IS NULL THEN  RETURN state;  END IF;  -- Конкатенируем состояние и значение через разделитель  RETURN state || delimiter || value;  END;  ';  -- Функция для возврата итогового результата (в данном случае просто возвращает результат state)  CREATE OR REPLACE FUNCTION concat\_with\_delimiter\_final(state TEXT)  RETURNS TEXT  LANGUAGE plpgsql AS  $$  BEGIN  RETURN state;  END;  $$;  -- Создание агрегатной функции  CREATE OR REPLACE AGGREGATE concat\_with\_delimiter(TEXT, TEXT) (  SFUNC = concat\_with\_delimiter\_agg, -- State transition function  STYPE = TEXT, -- Тип состояния (аккумулятор)  FINALFUNC = concat\_with\_delimiter\_final -- Final function  );  -- Пример таблицы  CREATE TEMP TABLE test\_data (  value TEXT  );  -- Заполнение тестовыми данными  INSERT INTO test\_data (value)  VALUES  ('123'),  ('456'),  ('789');  -- Использование агрегатной функции  SELECT concat\_with\_delimiter(value, '\_') AS concatenated\_result  FROM test\_data; |

5. Для всех ключевых полей (как первичных ключей, так и внешних) создать индексы командой CREATE INDEX.

|  |
| --- |
| /\*\*  Для всех ключевых полей (как первичных ключей, так и внешних) создать индексы командой CREATE INDEX.  \*/  CREATE INDEX ON electronic\_equipment (manufacturer);  CREATE INDEX ON furniture (material); |

Лабораторная работа №8

Реализация хронологической базы данных на реляционной платформе

**Цель**: научиться разрабатывать БД, поддерживающие свойство темпоральности данных.

Ход работы:

 Используя результаты выполнения лабораторной работы №7:

1. Разработать и реализовать базу данных, поддерживающую свойство темпоральности для данных, т.е. поддержка возможности отслеживания динамики изменения данных в БД, с возможностью перехода (отката) к любому предыдущему состоянию;
2. Разработать интерфейсы взаимодействия конечных пользователей с данными, поддерживающих возможность перехода (отката) к любому предыдущему состоянию БД.

Задание 1.

|  |
| --- |
| -- 1. Создаем функцию для получения состояния таблицы на определенный момент времени  CREATE OR REPLACE FUNCTION get\_table\_history(  table\_name TEXT,  moment TIMESTAMP DEFAULT now()  ) RETURNS SETOF RECORD AS $$  BEGIN  RETURN QUERY EXECUTE format('  SELECT \* FROM %I  WHERE time\_create <= %L  AND (time\_dead IS NULL OR time\_dead > %L)  ORDER BY id, time\_create',  table\_name, moment, moment);  END;  $$ LANGUAGE plpgsql;  -- 2. Создаем функцию для отката состояния объекта  CREATE OR REPLACE FUNCTION revert\_object(  target\_table TEXT,  object\_id INT,  revert\_to TIMESTAMP  ) RETURNS VOID AS $$  DECLARE  latest\_record RECORD;  revert\_record RECORD;  col\_name TEXT;  col\_list TEXT := '';  value\_list TEXT := '';  query\_text TEXT;  BEGIN  -- Находим последнюю версию объекта  EXECUTE format('  SELECT \* FROM %I  WHERE id = %s  ORDER BY time\_create DESC  LIMIT 1',  target\_table, object\_id) INTO latest\_record;    -- Если объект существует и не удален  IF latest\_record IS NOT NULL AND latest\_record.time\_dead IS NULL THEN  -- Устанавливаем время смерти текущей версии  EXECUTE format('  UPDATE %I  SET time\_dead = %L  WHERE id = %s AND time\_create = %L',  target\_table, now(), object\_id, latest\_record.time\_create);  END IF;    -- Находим версию для отката  EXECUTE format('  SELECT \* FROM %I  WHERE id = %s AND time\_create <= %L  AND (time\_dead IS NULL OR time\_dead > %L)  ORDER BY time\_create DESC  LIMIT 1',  target\_table, object\_id, revert\_to, revert\_to) INTO revert\_record;    -- Если нашли версию для отката  IF revert\_record IS NOT NULL THEN  -- Собираем список столбцов и значений для вставки  FOR col\_name IN  SELECT column\_name  FROM information\_schema.columns  WHERE table\_name = target\_table  AND column\_name NOT IN ('id', 'time\_create', 'time\_dead')  LOOP  IF col\_list <> '' THEN  col\_list := col\_list || ', ';  value\_list := value\_list || ', ';  END IF;    col\_list := col\_list || col\_name;    -- Получаем значение из revert\_record по имени столбца  EXECUTE format('SELECT ($1).%I::text', col\_name)  USING revert\_record  INTO value\_list;    -- Добавляем значение в список  value\_list := value\_list || quote\_literal(value\_list);  END LOOP;    -- Формируем и выполняем запрос на вставку  query\_text := format('  INSERT INTO %I (id, time\_create, time\_dead, %s)  VALUES (%s, %L, NULL, %s)',  target\_table, col\_list, object\_id, now(), value\_list);    EXECUTE query\_text;  END IF;  END;  $$ LANGUAGE plpgsql;  -- 3. Создаем представления для удобного доступа к истории изменений  CREATE OR REPLACE VIEW electronic\_equipment\_history AS  SELECT id, time\_create, time\_dead, warranty\_period, manufacturer,  CASE WHEN time\_dead IS NULL THEN 'Active' ELSE 'Inactive' END AS status  FROM electronic\_equipment  ORDER BY id, time\_create;  CREATE OR REPLACE VIEW furniture\_history AS  SELECT id, time\_create, time\_dead, material, dimensions,  CASE WHEN time\_dead IS NULL THEN 'Active' ELSE 'Inactive' END AS status  FROM furniture  ORDER BY id, time\_create;  CREATE OR REPLACE VIEW consumables\_history AS  SELECT id, time\_create, time\_dead, quantity, unit,  CASE WHEN time\_dead IS NULL THEN 'Active' ELSE 'Inactive' END AS status  FROM consumables  ORDER BY id, time\_create; |

Задание 2

|  |
| --- |
| -- 1. Функция для восстановления объекта на определенную дату (исправленная)  CREATE OR REPLACE FUNCTION restore\_object\_version(  target\_table TEXT,  object\_id INT,  version\_time TIMESTAMP  ) RETURNS TEXT AS $$  DECLARE  result TEXT;  col\_record RECORD;  col\_list TEXT := '';  value\_list TEXT := '';  query\_text TEXT;  BEGIN  -- Собираем список столбцов и значений для вставки  FOR col\_record IN  SELECT column\_name  FROM information\_schema.columns  WHERE table\_name = target\_table  AND column\_name NOT IN ('id', 'time\_create', 'time\_dead')  LOOP  IF col\_list <> '' THEN  col\_list := col\_list || ', ';  value\_list := value\_list || ', ';  END IF;    col\_list := col\_list || col\_record.column\_name;  value\_list := value\_list || format('(SELECT %I FROM %I WHERE id = %s AND time\_create <= %L ORDER BY time\_create DESC LIMIT 1)',  col\_record.column\_name,  target\_table,  object\_id,  version\_time);  END LOOP;    -- Формируем и выполняем запрос на вставку  query\_text := format('  INSERT INTO %I (id, time\_create, time\_dead, %s)  SELECT %s, now(), NULL, %s  WHERE EXISTS (  SELECT 1 FROM %I  WHERE id = %s AND time\_create <= %L  )',  target\_table, col\_list, object\_id, value\_list, target\_table, object\_id, version\_time);    EXECUTE query\_text;    -- Обновляем время смерти предыдущей активной версии  EXECUTE format('  UPDATE %I  SET time\_dead = now()  WHERE id = %s AND time\_dead IS NULL AND time\_create < now()',  target\_table, object\_id);    result := format('Object %s from table %I restored to version at %s',  object\_id, target\_table, version\_time);  RETURN result;  END;  $$ LANGUAGE plpgsql;  -- 2. Функция для сравнения двух версий объекта (исправленная)  CREATE OR REPLACE FUNCTION compare\_object\_versions(  target\_table TEXT,  object\_id INT,  version1 TIMESTAMP,  version2 TIMESTAMP  ) RETURNS TABLE (  column\_name TEXT,  value1 TEXT,  value2 TEXT,  changed BOOLEAN  ) AS $$  DECLARE  query\_text TEXT;  BEGIN  query\_text := format('  WITH  v1 AS (  SELECT \* FROM %I  WHERE id = %s AND time\_create <= %L  ORDER BY time\_create DESC LIMIT 1  ),  v2 AS (  SELECT \* FROM %I  WHERE id = %s AND time\_create <= %L  ORDER BY time\_create DESC LIMIT 1  )  SELECT  c.column\_name::TEXT,  COALESCE(  (SELECT jsonb\_path\_query\_first(  to\_jsonb(v1),  format(''$.%%s'', c.column\_name)::jsonpath  )::TEXT FROM v1),  ''N/A'') AS value1,  COALESCE(  (SELECT jsonb\_path\_query\_first(  to\_jsonb(v2),  format(''$.%%s'', c.column\_name)::jsonpath  )::TEXT FROM v2),  ''N/A'') AS value2,  EXISTS (  SELECT 1 FROM v1, v2  WHERE to\_jsonb(v1)->>c.column\_name IS DISTINCT FROM  to\_jsonb(v2)->>c.column\_name  ) AS changed  FROM information\_schema.columns c  WHERE c.table\_name = %L  AND c.column\_name NOT IN (''time\_create'', ''time\_dead'')  ORDER BY c.ordinal\_position',  target\_table, object\_id, version1,  target\_table, object\_id, version2,  target\_table);    RETURN QUERY EXECUTE query\_text;  END;  $$ LANGUAGE plpgsql; |

Дополнительно:

|  |
| --- |
| -- 1. Функция для просмотра истории объекта  CREATE OR REPLACE FUNCTION get\_object\_history(  target\_table TEXT,  object\_id INT  ) RETURNS TABLE (  id INT,  time\_create TIMESTAMP,  time\_dead TIMESTAMP,  status TEXT,  details JSONB  ) AS $$  BEGIN  RETURN QUERY EXECUTE format('  SELECT  id,  time\_create,  time\_dead,  CASE WHEN time\_dead IS NULL THEN ''Active'' ELSE ''Inactive'' END AS status,  to\_jsonb(t) - ''id'' - ''time\_create'' - ''time\_dead'' AS details  FROM %I t  WHERE id = %s  ORDER BY time\_create DESC',  target\_table, object\_id);  END;  $$ LANGUAGE plpgsql;  -- 2. Функция для получения состояния таблицы на момент времени  CREATE OR REPLACE FUNCTION get\_table\_history(  table\_name TEXT,  moment TIMESTAMP DEFAULT now()  ) RETURNS SETOF RECORD AS $$  BEGIN  RETURN QUERY EXECUTE format('  SELECT \* FROM %I  WHERE time\_create <= %L  AND (time\_dead IS NULL OR time\_dead > %L)  ORDER BY id, time\_create',  table\_name, moment, moment);  END;  $$ LANGUAGE plpgsql;  -- 3. Функция для восстановления версии объекта  CREATE OR REPLACE FUNCTION restore\_object\_version(  target\_table TEXT,  object\_id INT,  version\_time TIMESTAMP  ) RETURNS TEXT AS $$  DECLARE  result TEXT;  col\_record RECORD;  col\_list TEXT := '';  value\_list TEXT := '';  query\_text TEXT;  BEGIN  -- Собираем список столбцов и значений для вставки  FOR col\_record IN  SELECT column\_name  FROM information\_schema.columns  WHERE table\_name = target\_table  AND column\_name NOT IN ('id', 'time\_create', 'time\_dead')  LOOP  IF col\_list <> '' THEN  col\_list := col\_list || ', ';  value\_list := value\_list || ', ';  END IF;  col\_list := col\_list || col\_record.column\_name;  value\_list := value\_list || format('(SELECT %I FROM %I WHERE id = %s AND time\_create <= %L ORDER BY time\_create DESC LIMIT 1)',  col\_record.column\_name,  target\_table,  object\_id,  version\_time);  END LOOP;  -- Формируем и выполняем запрос на вставку  query\_text := format('  INSERT INTO %I (id, time\_create, time\_dead, %s)  SELECT %s, now(), NULL, %s  WHERE EXISTS (  SELECT 1 FROM %I  WHERE id = %s AND time\_create <= %L  )',  target\_table, col\_list, object\_id, value\_list, target\_table, object\_id, version\_time);  EXECUTE query\_text;  -- Обновляем время смерти предыдущей активной версии  EXECUTE format('  UPDATE %I  SET time\_dead = now()  WHERE id = %s AND time\_dead IS NULL AND time\_create < now()',  target\_table, object\_id);  result := format('Object %s from table %I restored to version at %s',  object\_id, target\_table, version\_time);  RETURN result;  END;  $$ LANGUAGE plpgsql;  -- 4. Функция для сравнения версий объекта  CREATE OR REPLACE FUNCTION compare\_object\_versions(  target\_table TEXT,  object\_id INT,  version1 TIMESTAMP,  version2 TIMESTAMP  ) RETURNS TABLE (  column\_name TEXT,  value1 TEXT,  value2 TEXT,  changed BOOLEAN  ) AS $$  DECLARE  query\_text TEXT;  BEGIN  query\_text := format('  WITH  v1 AS (  SELECT \* FROM %I  WHERE id = %s AND time\_create <= %L  ORDER BY time\_create DESC LIMIT 1  ),  v2 AS (  SELECT \* FROM %I  WHERE id = %s AND time\_create <= %L  ORDER BY time\_create DESC LIMIT 1  )  SELECT  c.column\_name::TEXT,  COALESCE(  (SELECT jsonb\_path\_query\_first(  to\_jsonb(v1),  format(''$.%%s'', c.column\_name)::jsonpath  )::TEXT FROM v1),  ''N/A'') AS value1,  COALESCE(  (SELECT jsonb\_path\_query\_first(  to\_jsonb(v2),  format(''$.%%s'', c.column\_name)::jsonpath  )::TEXT FROM v2),  ''N/A'') AS value2,  EXISTS (  SELECT 1 FROM v1, v2  WHERE to\_jsonb(v1)->>c.column\_name IS DISTINCT FROM  to\_jsonb(v2)->>c.column\_name  ) AS changed  FROM information\_schema.columns c  WHERE c.table\_name = %L  AND c.column\_name NOT IN (''time\_create'', ''time\_dead'')  ORDER BY c.ordinal\_position',  target\_table, object\_id, version1,  target\_table, object\_id, version2,  target\_table);  RETURN QUERY EXECUTE query\_text;  END;  $$ LANGUAGE plpgsql; |

Пример работы:

|  |
| --- |
| -- 1. Просмотр истории объекта  SELECT \* FROM get\_object\_history('furniture', 1);  -- 2. Восстановление версии объекта на определенную дату  -- Восстановление состояния на 25 марта 2025, 13:13:00 (версия с материалом "Wood")  SELECT restore\_object\_version('furniture', 1, '2025-03-25 13:13:00');  -- Проверяем результат  SELECT \* FROM get\_object\_history('furniture', 1) LIMIT 3;  -- 3. Сравнение двух версий объекта  SELECT \* FROM compare\_object\_versions(  'furniture',  1,  '2025-03-25 13:13:00',  '2025-04-15 12:26:00');  -- 4. Получение состояния таблицы на определенный момент времени  -- Получение состояния таблицы на 25 марта 2025, 13:13:30  SELECT \* FROM get\_table\_history('furniture', '2025-03-25 13:13:30') AS (  id INT,  time\_create TIMESTAMP,  time\_dead TIMESTAMP,  material VARCHAR(50),  dimensions VARCHAR(50)  ); |

Лабораторная работа №9

Базовые концепции документо-ориентированных СУБД

**Цель**: научиться работать с документо-ориентированными СУБД.

Ход работы:

Используя любую документо-ориентированную СУБД (CouchDB, MongoDB или др.), реализовать программное средство, демонстрирующее работу с данными, хранимыми в ДОСУБД.

**1. Структура базы данных**

СУБД: MongoDB (документо-ориентированная)

База данных: library

Коллекция: books

Схема документа (JSON)

Каждая книга представлена в виде документа со следующими полями:

{

"\_id": ObjectId("665a1b2c3d4e5f6a7b8c9d0e"), // Уникальный идентификатор

"title": "Война и мир", // Название книги

"author": "Лев Толстой", // Автор

"year": 1869, // Год издания

"genres": ["классика", "исторический роман"], // Жанры (массив)

"is\_available": true, // Доступность

"last\_borrowed": ISODate("2025-04-15T14:30:00Z") // Дата последнего взятия

}

**2. Схема работы:**

Пользователь → Приложение (Python) → Драйвер pymongo → MongoDB

**3. Структура приложения**

Язык: Python  
Драйвер для MongoDB: pymongo

Функционал приложения

1. Добавление книги (add\_book())
   * Запрашивает у пользователя: название, автора, год, жанры.
   * Автоматически устанавливает is\_available=True и last\_borrowed=null.
2. Поиск книг (find\_books())
   * Ищет по названию, автору или жанру (регистронезависимо).
   * Выводит полную информацию, включая ID.
3. Просмотр всех книг (list\_all\_books())
   * Сортировка по названию.
   * Отображает статус (🟢 Доступна / 🔴 На руках).
4. Обновление данных (update\_book())
   * Позволяет изменить любые поля (кроме \_id).
   * При изменении статуса обновляет last\_borrowed.
5. Удаление книги (`delete\_book())
   * Требует подтверждения.
   * Показывает данные книги перед удалением.

|  |
| --- |
| from pymongo import MongoClient  from datetime import datetime  from bson import ObjectId  from pprint import pprint # Для красивого вывода  # Подключение к MongoDB  client = MongoClient("mongodb://localhost:27017/")  db = client["library"]  books = db["books"]  def add\_book():  print("\n--- Добавление книги ---")  title = input("Название книги: ")  author = input("Автор: ")  year = int(input("Год издания: "))  genres = input("Жанры (через запятую): ").split(",")  book = {  "title": title,  "author": author,  "year": year,  "genres": [genre.strip() for genre in genres],  "is\_available": True,  "last\_borrowed": None  }  result = books.insert\_one(book)  print(f"\n✅ Книга добавлена (ID: {result.inserted\_id})")  def find\_books():  print("\n--- Поиск книг ---")  query = input("Поиск (название/автор/жанр): ")  results = books.find({  "$or": [  {"title": {"$regex": query, "$options": "i"}},  {"author": {"$regex": query, "$options": "i"}},  {"genres": {"$in": [query.strip()]}}  ]  })  print("\n🔍 Результаты поиска:")  for book in results:  status = "Доступна" if book.get("is\_available", True) else "На руках"  print(f"ID: {book['\_id']}")  print(f"Название: {book['title']}")  print(f"Автор: {book['author']}")  print(f"Год: {book['year']}")  print(f"Жанры: {', '.join(book['genres'])}")  print(f"Статус: {status}")  print("-" \* 30)  def list\_all\_books():  print("\n--- Все книги в библиотеке ---")  all\_books = books.find().sort("title", 1) # Сортировка по названию  if books.count\_documents({}) == 0:  print("В библиотеке пока нет книг.")  return  for book in all\_books:  status = "🟢 Доступна" if book.get("is\_available", True) else "🔴 На руках"  last\_borrowed = book.get("last\_borrowed", "")  if last\_borrowed:  last\_borrowed = last\_borrowed.strftime("%d.%m.%Y %H:%M")  print(f"\n📖 {book['title']} ({book['year']})")  print(f"👤 Автор: {book['author']}")  print(f"🏷️ Жанры: {', '.join(book['genres'])}")  print(f"📌 Статус: {status}")  print(f"ID: {book['\_id']}")  if last\_borrowed:  print(f"⏳ Последний раз брали: {last\_borrowed}")  print(f"🆔 ID: {book['\_id']}")  def update\_book():  print("\n--- Обновление книги ---")  list\_all\_books() # Показываем список для выбора  book\_id = input("\nВведите ID книги для обновления: ")  try:  book = books.find\_one({"\_id": ObjectId(book\_id)})  if not book:  print("❌ Книга не найдена!")  return  print(f"\nТекущие данные книги '{book['title']}':")  pprint(book)  updates = {}  print("\nВведите новые данные (оставьте пустым, если не меняется):")  title = input("Название: ")  if title: updates["title"] = title  is\_available = input("Доступна (y/n): ").lower()  if is\_available == "y":  updates["is\_available"] = True  updates["last\_borrowed"] = None  elif is\_available == "n":  updates["is\_available"] = False  updates["last\_borrowed"] = datetime.now()  books.update\_one({"\_id": ObjectId(book\_id)}, {"$set": updates})  print("\n✅ Книга обновлена")  except:  print("❌ Ошибка: неверный формат ID")  def delete\_book():  print("\n--- Удаление книги ---")  list\_all\_books() # Показываем список для выбора  book\_id = input("\nВведите ID книги для удаления: ")  try:  result = books.delete\_one({"\_id": ObjectId(book\_id)})  if result.deleted\_count > 0:  print("\n✅ Книга удалена")  else:  print("❌ Книга не найдена!")  except:  print("❌ Ошибка: неверный формат ID")  def main():  print("\n📚 Управление библиотекой (MongoDB)")  while True:  print("\n1. Добавить книгу")  print("2. Найти книги")  print("3. Показать все книги")  print("4. Обновить книгу")  print("5. Удалить книгу")  print("6. Выход")  choice = input("\nВыберите действие: ")  if choice == "1":  add\_book()  elif choice == "2":  find\_books()  elif choice == "3":  list\_all\_books()  elif choice == "4":  update\_book()  elif choice == "5":  delete\_book()  elif choice == "6":  print("\nДо свидания! 👋")  break  else:  print("❌ Неверный выбор")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**Контрольные вопросы:**

**1. Основные преимущества ДОСУБД и их применение**

Основные преимущества ДОСУБД включают:

* Повышение качества бизнес-аналитики благодаря консолидации данных из различных источников
* Ускорение выполнения запросов за счет специальной оптимизации для быстрой обработки данных
* Повышение качества данных через автоматическую очистку и согласование информации
* Возможность исторического анализа тенденций и прогнозирования sap.com

Наиболее целесообразно применять ДОСУБД в следующих случаях:

* Когда требуется объединение данных из различных источников
* При необходимости быстрой аналитики больших объемов данных
* В случаях, требующих исторического анализа и прогнозирования
* Для поддержки принятия бизнес-решений на основе данных

**2. Сущности хранения данных**

В современных ДОСУБД используются следующие сущности для хранения данных:

* Структурированные данные (информация о людях, продуктах, транзакциях)
* Неструктурированные данные (документы, изображения, видео, электронная почта)
* Метаданные (описывающие структуру и содержание хранимых данных)
* Информационные витрины (подразделения хранилища для конкретных отделов или направлений бизнеса)

**3. Формат хранения данных**

ДОСУБД поддерживает различные форматы хранения данных:

* Структурированный формат для организованных данных
* Полуструктурированный формат для документов и электронной почты
* Неструктурированный формат для медиафайлов и сырых данных

**4. Взаимодействие приложения с ДОСУБД**

Архитектура взаимодействия приложения с ДОСУБД включает три основных уровня:

*Уровень аналитики*

* Предоставляет инструменты для работы конечных пользователей
* Обеспечивает создание отчетов и аналитических панелей
* Позволяет отслеживать ключевые показатели эффективности (KPI)

*Семантический уровень*

* OLAP (Online Analytical Processing) серверы обеспечивают быструю аналитику данных
* OLTP (Online Transaction Processing) серверы обрабатывают оперативные транзакции
* Бизнес-метаданные описывают структуру и смысл хранимых данных

*Уровень данных*

* Центральная база данных хранит все данные
* ETL инструменты обеспечивают извлечение, преобразование и загрузку данных
* Озера данных дополнительно хранят необработанные данные для дальнейшего анализа

Для взаимодействия с ДОСУБД используются следующие языки и интерфейсы:

SQL для запросов к данным

API для интеграции с приложениями

Инструменты OLAP для аналитики

Интерфейсы ETL для загрузки и преобразования данных