

# 1. ЛАБОРАТОРНАЯ РАБОТА № 1

## «СОЗДАНИЕ ДИНАМИЧЕСКИХ БАЗ ДАННЫХ»

### 1.1. Цель работы

Изучение технологии подготовки и выполнения Пролог-программ в интегрированной среде, исследование способов организации динамических баз данных (БД) средствами языка Пролог.

### 1.2. Краткие теоретические сведения

#### 1.2.1 Введение в Пролог

Программа на языке Пролог состоит из фактов, правил и целевых утверждений. **Факт** представляет собой истинное утверждение. **Правило** представляет собой утверждение, которое истинно при определенных условиях. Совокупность фактов и правил образует *базу данных Пролога*.

Когда база данных загружена в память Пролог-системы, к ней можно обращаться с вопросами, формулируемыми в виде *целевых утверждений*.

Примеры простейших фактов, правил и целевых утверждений, а также основы работы в интегрированной среде изложены в методических указаниях [3].

#### 1.2.2. Объекты данных Пролога

Все объекты данных языка Пролог представляют собой **термы** [1, 2]. *Терм* может быть *константой*, *переменной* или *структурой* (составным термом). **Константы** подразделяются на *числа* и *атомы*. *Атомы* — это *символьные константы*, которые начинаются строчной буквой или заключаются в одинарные кавычки, или состоят из специальных символов. Примеры атомов: **сергей**, **'Сергей'**. **==>**.

*Имена переменных* начинаются с заглавных букв или символа подчеркивания “\_”. Областью действия переменной является *утверждение* (факт или правило). Переменные, которым присвоены значения, называются *конкретизированными*. Существуют *анонимные* переменные — переменные без имени. Они обозначаются символом подчеркивания. Каждая анонимная переменная уникальна, т.е. отличается от всех других анонимных переменных в утверждении.

**Структура** (составной терм) является объектом, состоящим из нескольких компонент. Она состоит из атома, который называется *функтором*, и *последовательности термов*. Например, **f(T1,T2,T3)** — структура, состоящая из функтора **f** и трёх компонент **T1,T2,T3** (термов). *Число компонент структуры* называется *арностью*.

Рассмотрим структуру, представляющую информацию о некотором сотруднике:

```
сотрудник( фео('Петренко','Сергей', 'Иванович'),
            рабочее_место('Отдел 4', 'инженер'),
            адрес(улица('Тенистая',7),город('Ростов')),
            телефон('68-23-42')).
```

Структура **сотрудник** содержит вложенные подструктуры (термы): **фio, рабочее\_место, адрес, телефон**. В свою очередь, подструктура **адрес** состоит из двух подструктур: **улица** и **город**. Таким образом, **все объекты данных Пролога** — это **термы**, компонентами которых являются другие термы [1,2].

Широко используемым **объектом данных** языка Пролог является список. **Список** состоит из **произвольного числа элементов**, заключаемых в квадратные скобки и разделяемых запятыми. Например: **[a, b, c, d]**. Для приведенного списка элемент **a** — это **голова** списка, а подсписок **[b, c, d]** — **хвост** списка.

Для представления списка в виде структуры, состоящей из головы и хвоста, в Прологе широко используется еще одно обозначение, в котором **голова и хвост списка** отделяются вертикальной чертой: **[Голова | Хвост]**.

### 1.2.3. Сопоставление (унификация) термов

**Сопоставление** — это процесс, на вход которого подаются два терма, а он проверяет, соответствуют ли эти термы друг другу. Если **термы не сопоставимы**, то процесс терпит **неудачу**. Если **термы сопоставимы**, то процесс **находит конкретизацию переменных**, **делающую эти термы тождественными**, и завершается **успешно**.

Возможность явного сопоставления двух термов проверяется с помощью оператора **"="**. Например, сопоставление

? – отец('Иван',P) = отец(O,'Сергей').

закончится успехом при следующей конкретизации переменных: **O='Иван'; P='Сергей'**.

В общем, два **терма сопоставляются по следующим правилам** [1]:

а) если термы **X** и **Y** — константы, то они **сопоставимы**, только **когда одинаковы**;

б) если терм **X** представлен **константой или структурой**, а терм **Y** — **не конкретизированной переменной**, то термы **X** и **Y** **сопоставимы**, и в **Y** **подставляется значение X**;

в) если термы **X** и **Y** — структуры, то они **сопоставимы**, когда у них **совпадают главные функторы и арность**, а также **сопоставимы соответствующие компоненты структуры**.

Поиск решений в пролог-системах протекает в автоматическом режиме с использованием принципа возврата к альтернативным вариантам возможных сопоставлений.

### 1.2.4. Управление выполнением пролог-программ

При рассмотрении пролог-программы полезно выделять **два уровня ее смысла**: декларативный и процедурный. **Декларативный смысл** пролог-программы связан с **отношениями**, объявленными (декларируемыми) в программе, он **определяет, достижимо ли целевое утверждение, и при каких значениях переменных**

оно будет верным. **Процедурный смысл** определяет, как пролог-система обрабатывает отношения пролог-программы, каким образом пролог-система отвечает на вопросы.

Рассмотрим правило **P:- Q, R**. Запятая между условиями **Q** и **R** обозначает **конъюнкцию**. Декларативная интерпретация правила может быть следующей: **P истинно, если условия Q и R истинны**. А **процедурный вариант** можно сформулировать так: чтобы решить задачу P, сначала реши подзадачу Q, а затем — подзадачу R. Таким образом, процедурная интерпретация фиксирует порядок, в котором обрабатываются подцели **Q** и **R**.

**Дизъюнкция** условий обозначается **точкой с запятой**. Например: **P:- Q ; R**. В этом случае пролог система должна оценить верность условия **Q** или R. Чтобы доказать **P** пролог-система сначала предпримет попытку доказательства **Q**. Если доказательство **Q** завершится неудачей (**fail**), то пролог-система вернется к точке выбора вариантов, удалит все сделанные подстановки и перейдет к доказательству альтернативного утверждения **R**. Однако она перейдет к доказательству **R** и в том случае, когда **Q** верно. Так как в процессе доказательства выполняются подстановки в переменные, то **доказательство альтернативного утверждения обеспечивает нахождение дополнительных вариантов подстановок**, которые могут интересовать пользователя.

Если проверку альтернативного условия требуется исключить, то применяют встроенный предикат **отсечения**, который обозначается знаком **“!**”. Этот предикат всегда выполняется успешно и **стирает все альтернативные ветви в пределах утверждения**, в котором он введен. Например: **P:- Q, ! ; R**. В этом случае при успешной попытке доказательства **Q** предикат отсечения сотрёт точку выбора альтернативного условия **R** и оно не будет проверяться.

Для управления процессом обработки условий также широко применяется встроенный **предикат fail**, который **обеспечивает создание состояния искусственной неудачи**. Состояние искусственной неудачи заставляет пролог-систему **возвращаться к имеющимся точкам выбора и искать другие варианты доказательства утверждений**. Точки выбора могут формироваться предикатами, допускающими альтернативные подстановки (такие предикаты называют **недетерминированными**), или создаваться искусственно с помощью специальных предикатов.

Таким специальным предикатом является **предикат repeat** [1,2]. Предикат определяется **рекурсивно**: **repeat:-true; repeat**. Благодаря этому он **создаёт бесконечное число точек выбора**. Его часто применяют совместно с **fail** для организации циклов, которые называются **циклами repeat-fail**. Общая схема организации таких циклов следующая:

**цикл:- repeat,**  
**(<проверка условия выхода из цикла>, !;**  
**<тело цикла>, fail).**

Здесь **скобки** образуют **группу** и тем самым **ограничивают область действия дизъюнкции**. Если условие завершения цикла выполняется **успешно**, то **предикат отсечения стирает все точки выбора и цикл завершается**. Иначе **выполняется тело**

цикла, а предикат **fail** обеспечивает возврат к точкам выбора, создаваемым с помощью **repeat**, и действия повторяются.

### 1.2.5. Способы представления базы данных

Существует несколько простых способов представления реляционных баз данных на языке Пролог [6]. Первый способ основан на представлении целостных информационных элементов, т.е. записей (кортежей) базы в виде множества фактов. Например:

```
%      N      Фам.      Имя      Отч.      Отд. Должн.  Филиал  Тел.
сотрудник(1001, петренко, сергей, иванович, 4, инженер, Ялта, 68-23-42).
```

Совокупность всех таких записей (кортежей) образует отношение **сотрудник**.

Второй способ состоит в представлении базы данных в виде множества фактов, которые устанавливают отношения между отдельными атрибутами записей. Один из атрибутов при этом должен выступать в роли ключа, объединяющего все остальные атрибуты:

```
фамилия(1001, петренко).
имя(1001, сергей).
отчество(1001, иванович).
отдел(1001,4).
должность(1001,инженер).
филиал(1001, ялта).
телефон(1001, 68-23-42).
```

Здесь в качестве ключевого атрибута используется табельный номер сотрудника. Все атрибуты сотрудника можно объединить в отношение **сотрудник** при помощи правила:

```
сотрудник(N, Фам, Имя, Отч, Отд, Должн, Филиал, Тел):-
    фамилия(N, Фам), имя(N, Имя), отчество(N, Отч),
    отдел(N, Отд), должность(N, Должн),
    филиал(N, Филиал), телефон(N, Тел).
```

Представление базы данных в виде отдельных атрибутов является более гибким, чем применение целостных записей, поскольку новые атрибуты можно добавлять без переписывания существующей базы данных.

Третий способ основан на представлении базы данных в виде списка структур. Каждый элемент списка — это отдельная запись базы данных:

```
%      N      Фам.      Имя      Отч.      Отд. Должн.  Филиал  Тел.
[сотр(1001, петренко, сергей, иванович, 4, инженер, ялта, 68-23-42),
сотр(1002, никулин, николай, васьильевич, 3, начальник, керчь, 11-24-47),
сотр(1003, павлов, сергей, николаевич, 2, оператор, керчь, 11-23-42)]
```

Если в первом и во втором способах база данных является частью программы, и для работы с ней в значительной степени используются встроенные поисковые механизмы пролог-системы, то в третьем случае база данных отделена от

программного кода и все поисковые процедуры для работы с такой базой должны определяться программистом самостоятельно.

### 1.2.6. Списки и рекурсия

Над списками выполняют следующие операции: добавление элемента в список, удаление элемента из списка, объединение списков, поиск элемента в списке др. Наиболее просто определяется добавление элемента в список [1]:

**добавить (X, L, [X|L]).**

Здесь **X** — добавляемый элемент; **L** — список, в который добавляется элемент; **[X|L]** — результирующий список. Таким образом, в ходе доказательства этого целевого утверждения, пролог-система добавит **X** в начало списка **L** и сформирует результирующий список **[X|L]**. Например, в результате доказательства цели **добавить (a, [b,c], Y)** будет получено **Y=[a,b,c]**.

Обработка списков часто выполняется с помощью рекурсивных предикатов. Например, определим отношение принадлежности — **принадлежит (X, L)**, где **X** — элемент, а **L** — список. Элемент **X** принадлежит списку **L**, если: 1) **X** есть голова списка **L**; 2) или **X** принадлежит хвосту списка **L**. Это можно записать в виде двух утверждений, первое из которых есть факт, а второе — правило:

**принадлежит(X, [X|Хвост]).**

**принадлежит(X, [Голова|Хвост]): – принадлежит(X,[Хвост]).**

В общем случае все такие определения обработки списков строятся по следующей схеме [1]:

**предикат(...[ ]...).**

**предикат(...[Голова|Хвост]...): – обработка(Голова), предикат(...[Хвост]...).**

Рекурсивные вызовы прекратятся, когда хвост списка окажется пустым списком. В приведенном определении первое утверждение определяет условие выхода из рекурсии, а второе утверждение — правило, предусматривающее при каждом вызове обработку очередного элемента списка и рекурсивный вызов определяемого предиката, аргументом которого является хвост списка.

### 1.2.7. Получение сведений из базы данных

Пролог является естественным языком запросов к реляционной базе данных. Пусть база данных представляется множеством фактов:

**сотрудник(N, Фам, Имя, Отч, Отд, Должн, Филиал, Тел).**

В запросах можно ссылаться на объекты данных, не указывая всех деталей. Например, проверка наличия в базе данных сведений о сотруднике **Петренко** может быть реализована запросом:

**?- сотрудник( \_, петренко, \_, \_, \_, \_, \_ ).**  
**true.**



**Символы подчеркивания** обозначают различные **анонимные переменные**, в которые могут **выполняться различные подстановки**. Чтобы извлечь из базы данных фамилии и телефоны сотрудников, следует в целевом утверждении (вопросе) на соответствующих позициях записать переменные:

?- **сотрудник**(\_, X, \_, \_, \_, \_, Y).  
 X = петренко,  
 Y = 68-23-42.

В этом случае пролог-система выполняет просмотр базы данных, находит терм **сотрудник** и сопоставляет X с фамилией, а Y — с номером телефона сотрудника.

Приведенные примеры запросов соответствуют **операции «проекция»** реляционной алгебры. В общем, *проекция* состоит в построении отношения, использующего лишь некоторые аргументы исходного n-арного отношения  $r(X_1, \dots, X_n)$ . Например, следующая проекция, записываемая в виде правила Пролога, оставляет первый и третий аргументы исходного отношения r

$r\_1\_3(X_1, X_3) :- r(X_1, \dots, X_n).$

Проекции широко используют при построении **отношений**, с помощью которых можно будет **выбирать конкретные компоненты структур**. Такие отношения можно назвать **селекторами**. Отношение-селектор будет иметь два аргумента: **первый аргумент** — **объект данных, содержащий компоненту**; **второй** — **переменная, представляющая извлекаемую компоненту**. Имя отношения-селектора будет совпадать с именем компоненты, которую нужно выбрать [2]:

**отношение-селектор(Объект, Компонента).**

Примеры некоторых отношений-селекторов для структуры **сотрудник**:

**фамилия**(**сотрудник**(\_, **Фамилия**, \_, \_, \_, \_), **Фамилия**).  
**должность**(**сотрудник**(\_, \_, \_, \_, **Должность**, \_), **Должность**).  
**телефон**(**сотрудник**(\_, \_, \_, \_, \_, **Телефон**), **Телефон**).

Также просто описывается любой конкретный случай **выборки**. Например, рассмотрим отношение, формирующие наборы данных, в которых второй аргумент отношения **сотрудник** — это Петренко или Павлов:

**один\_из\_двух\_сотрудников**(N, Фам, Имя, Отч, Отд, Должн, Филиал, Тел):-  
**сотрудник**(N, Фам, Имя, Отч, Отд, Должн, Филиал, Тел),  
 (Фам=петренко; Фам=павлов).

В этом примере пролог-система для очередной фамилии **Фам**, конкретизируемой при сопоставлении отношения **сотрудник** с фактами базы данных, проверяет, равна ли она Петренко или Павлов.

В том случае, когда **формируемые выборки** необходимо **собрать в список** для дальнейшей обработки удобно использовать встроенный предикат **bagof** [1,2]. Цель **bagof(X, C, L)** формирует **список L** всех **объектов X**, удовлетворяющих усло-

вию **С**. Например, требуется сформировать список **L**, представляющий подмножество сотрудников одного филиала:

**подмножество\_сотрудников(Филиал,L):-**

**bagof(сотрудник\_ф(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел),**  
**сотрудник(N,Фам,Имя,Отч,Отд,Должн,Филиал,Тел), L).**

Предикат формирует список **L** из новых отношений **сотрудник\_ф**. В дальнейшем при необходимости эти отношения можно добавить в базу данных.

Кроме проекции и выборки, выделяют еще 4 основные операции реляционной алгебры: объединение, пересечение, разность и декартово произведение.

Операция **объединения** двух  $n$ -арных отношений  $r_1(X_1, \dots, X_n)$  и  $r_2(X_1, \dots, X_n)$  строит **новое  $n$ -арное отношение**, содержащие **множество кортежей, принадлежащих либо первому отношению, либо второму**. Новое отношение задается на Прологе следующим правилом:

**объединение\_r1\_r2( $X_1, \dots, X_n$ ):-  $r_1(X_1, \dots, X_n); r_2(X_1, \dots, X_n)$ .**

Например, если имеются два отношения **сотрудник**, которые содержат сведения о сотрудниках двух филиалов, то объединенное отношение будет содержать общий перечень сотрудников.

**Пересечением** называется отношение, которое содержит **множество кортежей, принадлежащих одновременно и первому и второму** отношениям:

**пересечение\_r1\_r2( $X_1, \dots, X_2$ ):-  $r_1(X_1, \dots, X_n), r_2(X_1, \dots, X_n)$ .**

Например, применительно к базе данных сотрудников двух филиалов с помощью пересечения можно получить список сотрудников, занимающих одну и ту же должность.

**Разностью** двух отношений  $r_1$  и  $r_2$  называется отношение, содержащее множество кортежей, **принадлежащих  $r_1$  и не принадлежащих  $r_2$** :

**разность\_r1\_r2( $X_1, \dots, X_2$ ):-  $r_1(X_1, \dots, X_n), \text{not } r_2(X_1, \dots, X_n)$ .**

**Декартово произведение** может быть определено следующим правилом. Если  $r$   $m$ -арное отношение, а  $s$   $n$ -арное, то  $(m+n)$ -арное отношение, представляющее декартово произведение, определится так:

**декартово\_произведение\_r\_s( $X_1, \dots, X_{m+n}$ ):-  $r(X_1, \dots, X_m), s(X_{m+1}, \dots, X_{m+n})$ .**

Операция декартова произведения обычно используется для генерации некоего **отношения**, которое характеризует **все возможные комбинации между кортежами отношений  $r$  и  $s$** . В дальнейшем из этого отношения можно получать различные проекции.

### 1.2.8. Предикаты для работы с динамической базой данных

В Прологе имеется ряд **встроенных предикатов**, которые позволяют изменять базу данных пролог-системы в процессе выполнения программ: **assert(X)**, **asserta(X)**, **assertz(X)**, **retract(X)** [1, 2, 10].

Предикат **assert(X)** добавляет в базу данных утверждение **X**. Например:

? – **assert(столица('Египет', 'Каир'))**.

Yes

? – **столица('Египет', X)**.

**X = Каир**

Предикат **asserta(X)** добавляет утверждение **X** в начало базы данных, предикат **assertz(X)** — в конец базы данных. Предикат **retract(X)** выполняет поиск утверждения **X** в базе данных и удаляет первое найденное утверждение. Встроенный предикат **retractall(X)** удаляет из базы данных все утверждения, функтор и аргументы которых сопоставимы с **X**.

Для просмотра утверждений, входящих в базу данных, можно воспользоваться предикатами: **listing** — выводит утверждения, содержащиеся в базе данных, в выходной поток; **listing(X)** — печатаются утверждения, сопоставимые с **X**.

### 1.2.9. Предикаты ввода-вывода

Пролог взаимодействует с *входными и выходными потоками*. В каждый момент времени Пролог взаимодействует с одним входным и одним выходным потоками. Перечень встроенных предикатов, предназначенных для управления входными и выходными потоками, приведен в таблице 1.1. Для считывания значений из входных потоков и записи их в выходные потоки применяют следующие предикаты:

**read(X)** — вызывает чтение очередного термина из входного потока и сопоставление его с **X**, вводимые термины должны заканчиваться точкой, в конце файла **X** конкретизируется атомом **end\_of\_file**;

**write (X)** — выводит терм **X** в текущий выходной поток;

**nl** — переход на новую строку;

**tab(N)** — выводит в выходной поток **N**-пробелов;

**put(X)** — выводит символ с ASCII кодом **X** на терминал;

**get0(X)** — считывает один символ с клавиатуры и сопоставляет его **X**;

**get(X)** — сопоставляет **X** с первым символом, отличным от пробела;

**display(X)** — выводит терм **X** в стандартной скобочной записи в текущий выходной поток.

Таблица 1.1— Предикаты управления потоками

Входной поток	Выходной поток	Интерпретация предиката
<b>see(&lt;имя файла&gt;)</b>	<b>tell(&lt;имя файла&gt;)</b>	Определяет в качестве текущего входного или выходного потока соответствующий файл.
<b>seeing(&lt;имя файла&gt;)</b>	<b>telling(&lt;имя файла&gt;)</b>	Отождествляет имя файла с текущим входным или выходным потоком.
<b>seen</b>	<b>told</b>	Закрывает текущий входной или выходной поток и опять связывает с текущим входным или выходным потоком поль-



	зовательский терминал.
--	------------------------

Файлы Пролога являются текстовыми и обрабатываются последовательно. Типовая последовательность вызова встроенных предикатов при выполнении ввода из файла является следующей:

<code>see('имя файла'),</code>	<code>% открытие файла</code>
<code>read(X),</code>	<code>% чтение термина X</code>
<code>...,</code>	<code>% обработка</code>
<code>seen.</code>	<code>% закрытие файла</code>

Аналогично строится работа с выходным потоком. Примеры применения предикатов ввода-вывода приведены в приложении Б методических указаний [3].

### 1.3. Варианты заданий

Написать программу, обеспечивающую создание динамической базы данных. Структура базы данных определяется одной из таблиц в соответствии с вариантом задания. В функции программы должно входить:

- добавление записи в базу данных;
- удаление записи из базы данных;
- просмотр базы данных;
- сохранение базы данных в файле;
- загрузка базы данных из файла;
- реализация операций реляционной алгебры (на примерах).

Кроме этого, программа должна выполнять дополнительные функции, указанные в варианте задания (таблица 1.2).

Таблица 1.2 — Варианты заданий

Вариант	Номер таблицы и дополнительные функции
1.	Таблица 1.3. Корректировка данных в базе по номеру записи; вывод на дисплей фамилий и номеров групп для всех студентов, если средний балл студента больше 4.0; если таких студентов нет, вывести соответствующее сообщение.
2.	Таблица 1.3. Корректировка данных в базе по фамилии; вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5; если таких студентов нет, вывести соответствующее сообщение.
3.	Таблица 1.3. Корректировка данных в базе по номеру группы; вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2; если таких студентов нет, вывести соответствующее сообщение.
4.	Таблица 1.4. Корректировка данных в базе по номеру рейса; вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение.
5.	Таблица 1.4. Корректировка данных в базе по типу самолета; вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение.
6.	Таблица 1.5. Корректировка данных в базе по фамилии; вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры; если таких работников нет, вывести на дисплей соответствующее сообщение.

Вариант	Номер таблицы и дополнительные функции
7.	Таблица 1.6. Корректировка данных в базе по номеру поезда; вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени; если таких поездов нет, выдать на дисплей соответствующее сообщение.
8.	Таблица 1.6. Корректировка данных в базе по пункту назначения; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение.
9.	Таблица 1.6. Корректировка данных в базе по времени отправления; вывод на экран информации о поезде, номер которого введен с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение.
10.	Таблица 1.7. Корректировка данных в базе по начальному маршруту; вывод на экран информации о маршруте, номер которого введен с клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.
11.	Таблица 1.7. Корректировка данных в базе по номеру маршрута; вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.
12.	Таблица 3.8. Корректировка данных в базе по фамилии; вывод на экран информации о человеке, номер телефона которого введен с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.
13.	Таблица 1.8. Корректировка данных в базе по номеру телефона; вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.
14.	Таблица 1.8. Корректировка данных в базе по году рождения; вывод на экран информации о человеке, чья фамилия введена с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.
15.	Таблица 1.9. Корректировка данных в базе по фамилии; вывод на экран информации о человеке, чья фамилия введена с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение.
16.	Таблица 1.9. Корректировка данных в базе по знаку зодиака ; вывод на экран информации о людях, родившихся под знаком, название которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.
17.	Таблица 3.9. Корректировка данных в базе по месяцу рождения ; вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры; если таких нет, выдать на дисплей соответствующее сообщение.
18.	Таблица 1.10. Корректировка данных в базе по названию товара; вывод на экран информации о товаре, название которого введено с клавиатуры; если таких товаров нет, выдать на дисплей соответствующее сообщение.
19.	Таблица 1.10. Корректировка данных в базе по названию магазина; вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры; если такого магазина нет, выдать на дисплей соответствующее сообщение.
20.	Таблица 1.11. Корректировка данных в базе по расчетному счету плательщика ; вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры; если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.
21.	Таблица 1.12 Корректировка данных в базе по фамилии; вывод анкетных данных студентов отличников; если таких студентов нет, вывести соответствующее сообщение.

Продолжение таблицы 1.2

Вариант	Номер таблицы и дополнительные функции
22.	Таблица 1.12. Корректировка данных в базе по году рождения; вывод на дисплей анкетных данных студентов, получивших одну оценку 3; если таких студентов нет, вывести соответствующее сообщение.
23.	Таблица 1.12. Корректировка данных в базе по году поступления; вывод на дисплей анкетных данных студентов, получивших все двойки; если таких студентов нет, вывести соответствующее сообщение.
24.	Таблица 1.12. Корректировка данных в базе по оценке «физика»; вывод на дисплей анкетных данных студентов, получивших все пятерки; если таких студентов нет, вывести соответствующее сообщение.
25.	Таблица 1.12. Корректировка данных в базе по номеру ; вывод на дисплей анкетных данных студентов, получивших одну оценку 4, а все остальные – 5; если таких студентов нет, вывести соответствующее сообщение.
26.	Таблица 1.12. Корректировка данных в базе по фамилии, которая начинается с литеры 'А' ; вывод на дисплей фамилий студентов, которые начинаются с литеры 'А', и их оценки; если таких студентов нет, вывести соответствующее сообщение.
27.	Таблица 1.12. Корректировка данных в базе по фамилии, которая начинается с литеры 'Б'; вывод на дисплей фамилий студентов, которые начинаются с литеры 'Б', и год их рождения; если таких студентов нет, вывести соответствующее сообщение.
28.	Таблица 1.12. Корректировка данных в базе по фамилии, которая начинается с литеры 'А'или 'Д' ; вывод на дисплей фамилий студентов, которые начинаются с литеры 'А'или 'Д', и год их поступления; если таких студентов нет, вывести соответствующее сообщение.

Таблица 1.3. — Студент группы

Фамилия И.О.	Номер группы	Успеваемость				
		P1	P2	P3	P4	P5

Таблица 1.4. — Рейс самолета

Пункт назначения	Номер рейса	Тип самолета
------------------	-------------	--------------

Таблица 1.5. — Сотрудник

Фамилия И.О.	Должность	Год приема на работу
--------------	-----------	----------------------

Таблица 1.6. — Поезд

Пункт назначения	Номер поезда	Время отправления
------------------	--------------	-------------------

Таблица 1.7. — Маршрут

Начальный пункт	Конечный пункт	Номер маршрута
-----------------	----------------	----------------

Таблица 1.8. — Записная книжка

Фамилия Имя	Номер телефона	Дата рождения		
		день	месяц	год

Таблица 1.9. — Знак зодиака

Фамилия Имя	Знак зодиака	Дата рождения		
		день	месяц	год

Таблица 1.10. — Стоимость

Название товара	Название магазина	Стоимость товара, грн
-----------------	-------------------	-----------------------

Таблица 1.11. — Счет

Расчетный счет плательщика	Расчетный счет получателя	Перечисляемая сумма, грн.
----------------------------	---------------------------	---------------------------

Таблица 1.12. — Студент

Номер	Фамилия Имя	Год рождения	Год поступления	Оценки		
				Ф	ВМ	Пр.

## 1.4. Порядок выполнения лабораторной работы

1.4.1. Изучить среду программирования по инструкции в МУДЛ или с использованием материалов из [3]. ~~Выполнить все приведенные примеры~~ в окне консоли среды программирования.

1.4.2. Ознакомиться по лекционному материалу или книгам [1, 2] с объектами данных и сопоставлением в языке Пролог, организацией управления в пролог-программах, предикатами обработки списков, встроенными предикатами работы с базой данных и предикатами ввода-вывода. Изучить примеры применения этих предикатов, приведенные в разделе 1.2 настоящей лабораторной работы.

1.4.2. Ознакомиться с вариантом задания и выбрать один из способов для хранения записей базы данных (см. п. 1.2.5).

1.4.3. Ознакомиться с примером кода, приведенного в приложении А, и по аналогии определить на языке Пролог для заданного варианта предикаты добавления записи в базу, удаления записи, просмотра базы, сохранения базы в файле и загрузки её из файла.

1.4.4. Создать в среде программирования Пролог-проект в соответствии с инструкцией в МУДЛ по использованию SWI-Prolog или методическими указаниями [3], содержащий подготовленные определения предикатов, указанных в п. 1.4.3.

1.4.5. Выполнить частичную отладку проекта.

1.4.6. Разработать определения дополнительных предикатов выборки и корректировки записей базы данных в соответствии с вариантом. При этом выборку записей из базы выполнять путем реализации операции проекции реляционной алгебры, следуя общим рекомендациям, указанным в п. 1.2.7.

1.4.7. Разработать предикаты, реализующие примеры операций реляционной алгебры (объединение, пересечение, разность) в соответствии с п. 1.2.7.

1.4.8. Выполнить полную отладку проекта и зафиксировать результаты работы программы в виде экранных копий.

## 1.5. Содержание отчета

Цель работы, вариант задания, обоснование выбранного представления базы данных, описание определений предикатов для общей работы с базой данных, описание разработанных дополнительных предикатов в соответствии с вариантом

задания, описание примеров реализации операций реляционной алгебры, тестовых запросов и результатов их выполнения, выводы.

## 1.6. Контрольные вопросы

- 1.6.1. Назовите составные части пролог-программы и приведите их определения.
- 1.6.2. Сформулируйте определения следующих понятий языка Пролог: терм, константа, атом, переменная, анонимная переменная, структура, список.
- 1.6.3. Что понимают под сопоставлением (унификацией) в языке Пролог?
- 1.6.4. Сформулируйте правила сопоставления термов в Прологе.
- 1.6.5. В чем заключаются декларативный и процедурный смыслы пролог-программы?
- 1.6.6. Объясните процедурный смысл простейших правил с конъюнкцией и дизъюнкцией условий.
- 1.6.7. Объясните назначение предиката отсечения. Приведите пример.
- 1.6.8. С какой целью применяется встроенный предикат **fail** ?
- 1.6.9. Приведите определение цикла **repeat-fail** .Объясните порядок его выполнения.
- 1.6.10. Объясните понятие группа в языке Пролог.
- 1.6.11. Назовите и объясните простейшие способы представления базы данных с помощью языка Пролог.
- 1.6.12. Определите следующие предикаты для работы со списками **L**: **добавить(X,L)**, **принадлежит(X,L)**, **объединить(L1,L2,L3)**.
- 1.6.13. Определите операцию “проекция” в виде правила Пролога. Покажите, как с помощью этой операции реализуются отношения-селекторы.
- 1.6.14. Объясните встроенный предикат **bagof(X, C, L)**.
- 1.6.15. Определите следующие операции реляционной алгебры в виде правил Пролога: объединение, пересечение, разность, декартово произведение.
- 1.6.16. Перечислите и объясните встроенные предикаты добавления утверждений в базу данных Пролога.
- 1.6.17. Назовите и объясните встроенные предикаты для удаления утверждений из базы данных Пролога.
- 1.6.18. Как просмотреть утверждения базы данных Пролога?
- 1.6.19. Что понимают под потоками ввода-вывода? Как открыть поток?
- 1.6.20. Какие предикаты используют для записи термов в поток?
- 1.6.21. Какие предикаты используют для чтения термов из потока?
- 1.6.22. Какие предикаты применяются для ввода-вывода символов?
- 1.6.23. Какое значение получает считываемый терм при достижении метки конец файла?