Логическое программирование

Занятие 3 / 03.05.22

Visual Prolog. Создание проекта

Project – New… - окно создания нового проекта

Имя проекта – так же как и `.exe` файл

Тип: `gui/console` (пока выбираем `console`)

Базовая директория – база всех проектов

Поддиректория – как и имя проекта

При нажатии ОК начинается сборка пустого проекта. В окне **Message** отображается процесс сборки

Visual Prolog. Сборка

В меню **Build** можно скомпилировать и запустить проект.

При запуске консольного проекта происходит выполнение цели в консоли; при запуске проекта с GUI открывается окно, похожее на сам Visual Prolog

Build – сборка измененных файлов проекта

Rebuild All – пересобрать весь проект

Execute – запустить проект (**.** exe файл)

Run in Window – запустить проект и оставить консольное окно открытым (выбираем этот пункт, чтобы видеть результаты)

Visual Prolog. Содержимое проекта

В окне проекта – дерево файлов проекта

`$(ProDir)` указывает на директорию, в которой установлен Visual Prolog. Она содержит необходимые библиотеки

`.ph` – заголовки пакетов. Пакет – набор классов и интерфейсов, которые будут использоваться

`.pack` – сами пакеты. Содержат определение или реализацию соответствующих файлов `.ph`

`.i` – интерфейсы

`.cl` – объявления классов

`.pro` – содержат реализацию классов (основной файл, где пишем код)

Visual Prolog vs Prolog

Структура программного кода

Файловая структура

Области видимости

Объектная ориентированность

Visual Prolog. Структура программного кода

Прежде чем определять предикат, его необходимо объявить

Прежде чем использовать домены, их необходимо определить

Вся программа (код класса) разбивается на блоки, которые следуют друг за другом. Блок заканчивается там, где начинается следующий блок

Блоки могут повторяться (может быть несколько однотипных блоков)

Visual Prolog. Структура программного кода

```
implement main
    open core

constants
    ...

class facts
    ...

clauses
    ...

class predicates
    ...

clauses
    ...

end implement main
```

Visual Prolog. Структура программного кода

- **`implement`** и **`end implement`** – ограничивают код некоторого класса. Имя класса ставится после `implement`
- **`open`** – используется для расширения области видимости класса. Ставится сразу после `implement`
- **`constants`** – блок констант. Имена констант пишутся с маленькой буквы. В конце ставится точка

Visual Prolog. Структура программного кода

```
implement main
    open core, stdio

constants
    faculty = "Физико-
математических и естественных наук".

clauses
    run() :-
        write(faculty).

end implement main
```

Visual Prolog. Структура программного кода

- **`domains`** – раздел объявления доменов (псевдонимов типов, собственных типов, структур данных)
- **`class facts`** – раздел объявления фактов, которые будут использованы в коде программы. Факты объявляются при помощи имени и аргументов в соответствии с разделом `domains`
- **`class predicates`** – раздел объявления предикатов (правил). Объявляются аналогично фактам

Visual Prolog. Структура программного кода

```
domains
    gender = male; female.

class facts
    person : (integer Id, string Name, gender Gender).
    parent : (integer ChildId, integer ParentId).

class predicates
    father : (string ChildName, string FatherName) nondeterm
anyflow.
```

Visual Prolog. Структура программного кода

```
class facts
    <name> : (<dtype> Arg1, …, <dtype> ArgN) <ptype>.

class predicates
<name> : (<dtype> Arg1, …, <dtype> ArgN) <ptype> <flow>.
```

Visual Prolog. Структура программного кода

<name> – символическое имя

<dtype> – тип данных аргумента (встроенный, например, `integer` или `string`, или определенный в разделе `domains`)

<ptype> – тип согласованности предиката с базой знаний. Определяет, сколько раз данный факт/предикат может быть согласован с базой знаний

<flow> – поточность аргументов предиката. Определяет, какие из аргументов являются входными, а какие выходными при попытке доказательства предиката

Visual Prolog. Структура программного кода

<ptype> для **class facts**

nondeterm (по умолчанию)

determ

single

Visual Prolog. Структура программного кода

<ptype> для **class facts**

nondeterm (по умолчанию) – согласуется с БЗ 0 и более раз

determ – согласуется с БЗ 0 или 1 раз

single – согласуется с БЗ строго 1 раз

Visual Prolog. Структура программного кода

<ptype> для **class predicates**

procedure (по умолчанию)

nondeterm

determ

multi

failure

Visual Prolog. Структура программного кода

<ptype> для **class predicates**

procedure (по умолчанию) – 1 раз (всегда истинен, не порождает новых решений)

nondeterm – 0 и более раз

determ – 0 или 1 раз

multi – 1 и более раз (всегда истинен, может порождать новые решения)

failure – 0 раз (никогда не согласуется с БЗ)

Visual Prolog. Структура программного кода

<flow> для **class predicates**

anyflow – любой аргумент может быть входным или выходным

(<i|o>, <i|o>, …, <i|o>) – указывает **i** (входной) или **o** (выходной) для каждого аргумента

Visual Prolog. Структура программного кода

```
class predicates
    father : (string ChildName, string FatherName) nondeterm (i,
o) (o, i) (o, o).

    father : (string ChildName, string FatherName) determ (i, i).

    ancestor : (string Person, string Ancestor) nondeterm (i, o).

    printPeople : (). /* procedure */
```

Visual Prolog. Структура программного кода

<ptype> для предиката определяется предикатами его тела

Обычно он соответствует «наиболее широкому» типу, который встречается в теле правила

Visual Prolog. Структура программного кода

Пусть есть набор предикатов с соответствующими типами:

```
p() – procedure
d() – determ
m() – multi
f() – failure
n() – nondeterm
```

Visual Prolog. Структура программного кода

Каких типов будут следующие предикаты (сколько раз они могут согласоваться с БЗ):

```
pred() :- p1(), p2().
pred() :- n(), p().
pred() :- p1(), d(), p2().
pred() :- d(), p(), m().
pred() :- p(), f().
pred() :- d(), n(), m(), p().
pred() :- n(), f().
```

Visual Prolog. Структура программного кода

- **`clauses`** – содержит определения ранее объявленных предикатов. Основной раздел программы (так же, как и в интерпретаторе)
- **`goal`** – определяет главную точку входа в программу; отсюда начинается выполнение

Visual Prolog. Структура программного кода

```
implement main
    open core, stdio

constants
    hello = "Hello, world!".

clauses
    run() :- write(hello).

end implement main

goal
    console::runUtf8(main::run).
```

Visual Prolog. Файловая структура

Visual Prolog обладает возможностью написания различных фрагментов программы в различных файлах. Данный процесс предоставляется и контролируется IDE

Примером может служить многократное использование некоторого домена в различных модулях программы

Visual Prolog. Области видимости

Предикат **p1** не определен в текущем классе, но определен в классе **class1**. Тогда его можно использовать двумя способами:

```
clauses
  p2 :- …
  p3 :- class1::p1, p2, …
```

```
open class1
…
clauses
  p2 :- …
  p3 :- p1, p2, …
```

Visual Prolog. Дополнительные фишки

Именованная база фактов. Позволяет обращаться к БФ по имени и загружать/выгружать ее содержимое в файл

```
class facts – databaseName
```

retractFactDB(<dbName>) – удаляет из динамической БЗ указанную БФ

file::consult(<filename>, <dbName>) – консультирует из файла БЗ, загружая указанную БФ

Visual Prolog

Программа, описывающая родственные отношения, с загрузкой БФ из внешнего файла

Преобразование базы знаний

assert(<predicate>) – добавляет предикат в динамическую базу знаний. Имеет тип `procedure`.

asserta(<predicate>), assertz(<predicate>) – добавляют в начало и в конец БЗ

retract(<predicate>) – удаляет предикат из динамической базы знаний. Имеет тип `nondeterm`. Удаление проходит успешно в том случае, если в БЗ нашлась цель, которую удалось унифицировать с `predicate`.

Visual Prolog

Программа увеличения стипендии

Visual Prolog. Циклы при помощи `fail`

В Prolog нельзя изменять значения переменной, если она уже была конкретизирована. Выражения вида `C = C + 1` не будут работать

Введем факт типа `single` или `determ`, который будет играть роль внешней переменной

Будем перезаписывать ее значения при помощи `assert`

Предикат, реализующий цикл, разобьем на 2 части:

- Часть, ответственная за перебор (обычно типа `fail`)
- Часть, ответственная за сбор результатов (обычно – `procedure`)

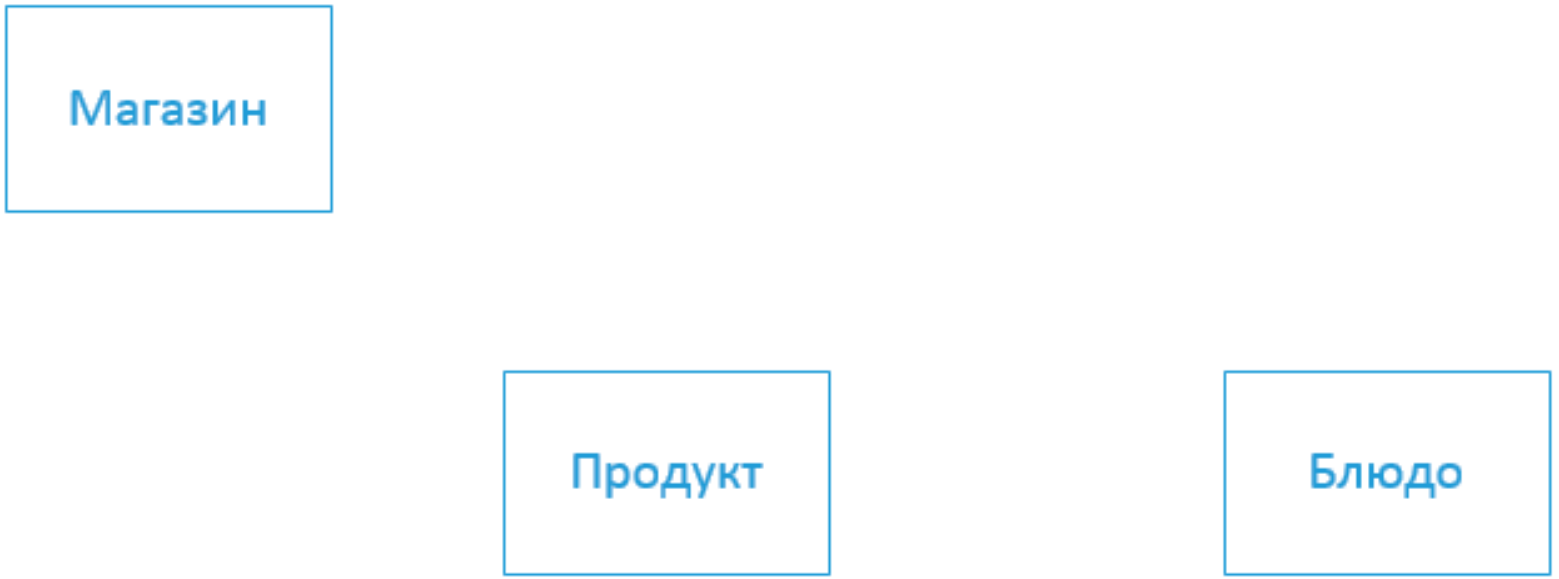
Visual Prolog. Циклы при помощи `fail`

Программа подсчета количества определенного факта в БЗ

Программа накопления суммы

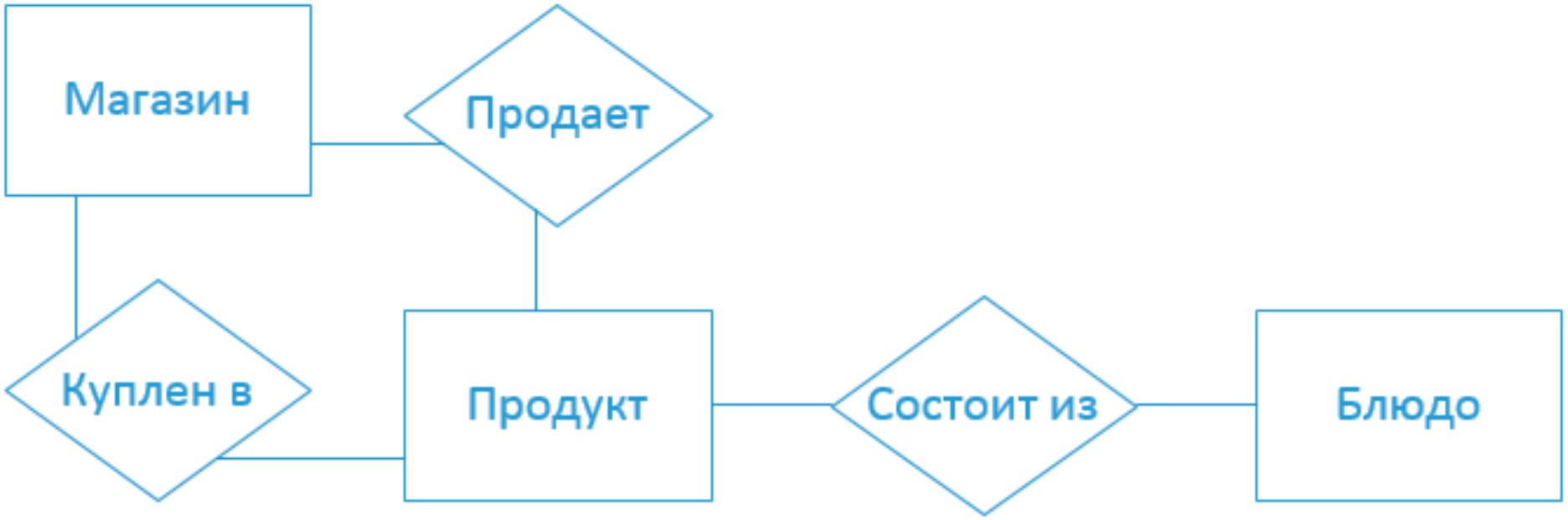
Проектирование предметной области

Составляем схему предметной области. Выделяем ключевые объекты (сущности)



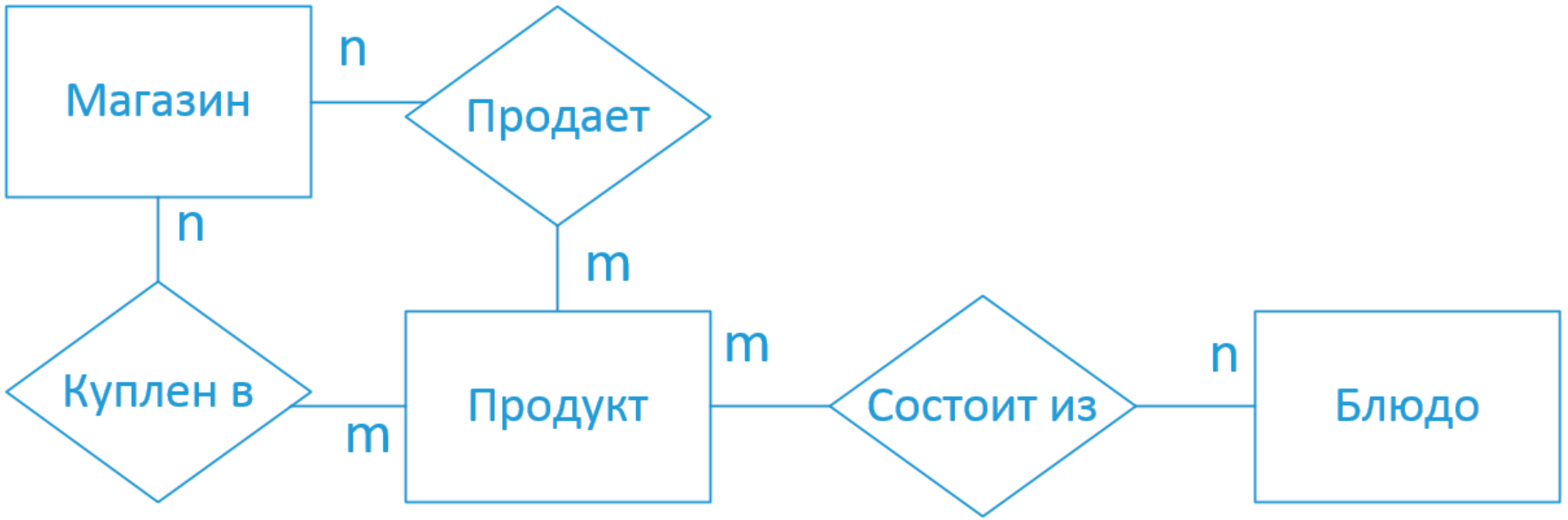
Проектирование предметной области

Выделяем отношения между объектами

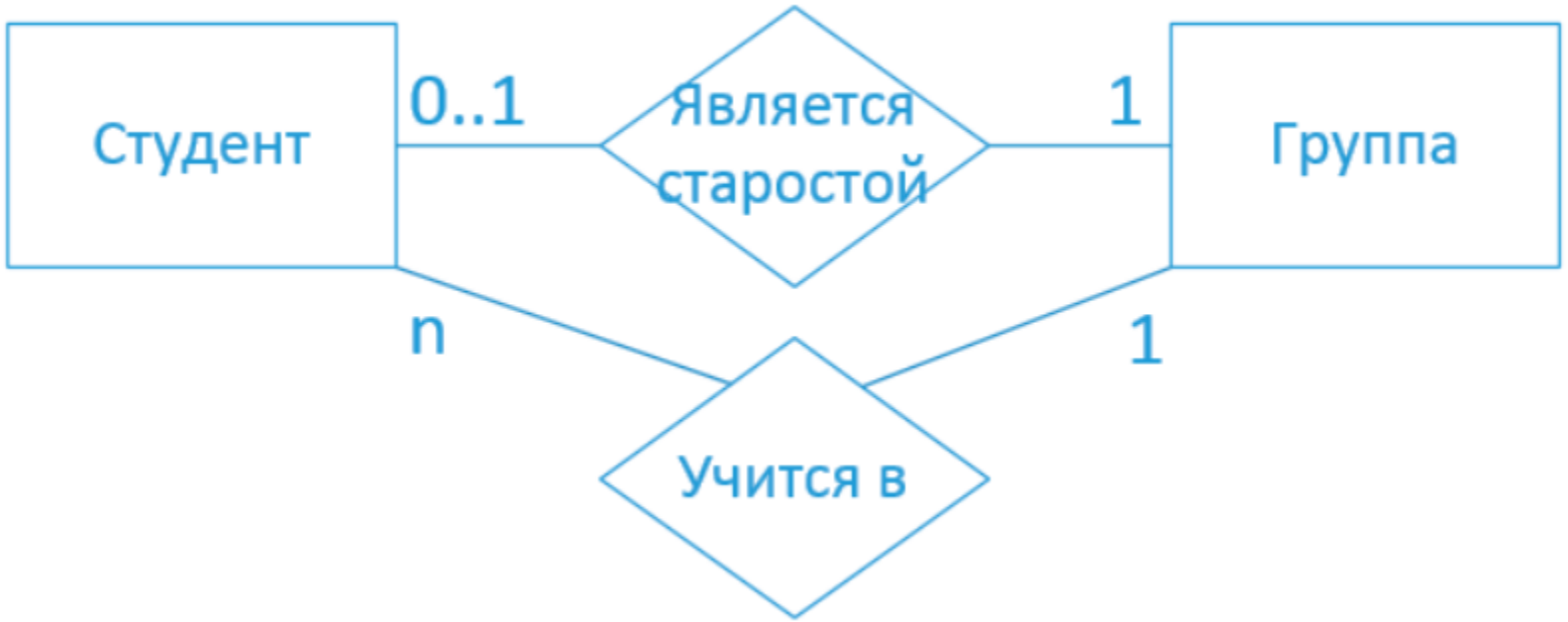


Проектирование предметной области

Устанавливаем кардинальности (типы отношений)



Проектирование предметной области



Проектирование предметной области

Реализация кардинальностей:

1 – 0..1 : добавляется внешний ключ (ссылка на объект) в объект со стороны 1

1 – N : добавляется внешний ключ в объект со стороны N

N – M : связь разбивается. Создается новая сущность, представляющая собой связь двух объектов. Каждый из объектов связывается с новой сущностью как 1 – N.

Проектирование предметной области

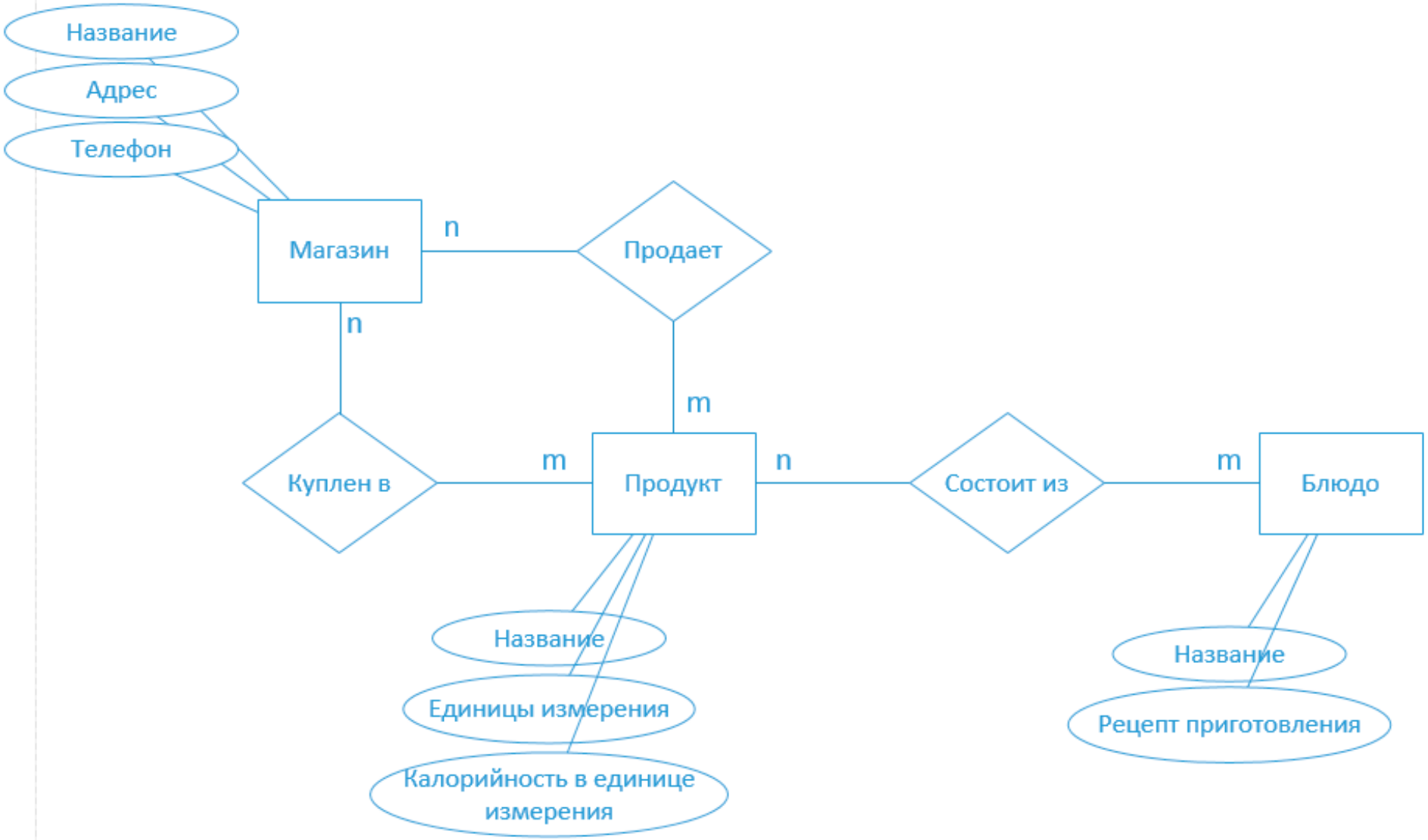
студент(Студ. билет, Имя, Дата рождения, Номер группы).

группа(Номер группы, Название, Год поступления, Студ. билет старосты).

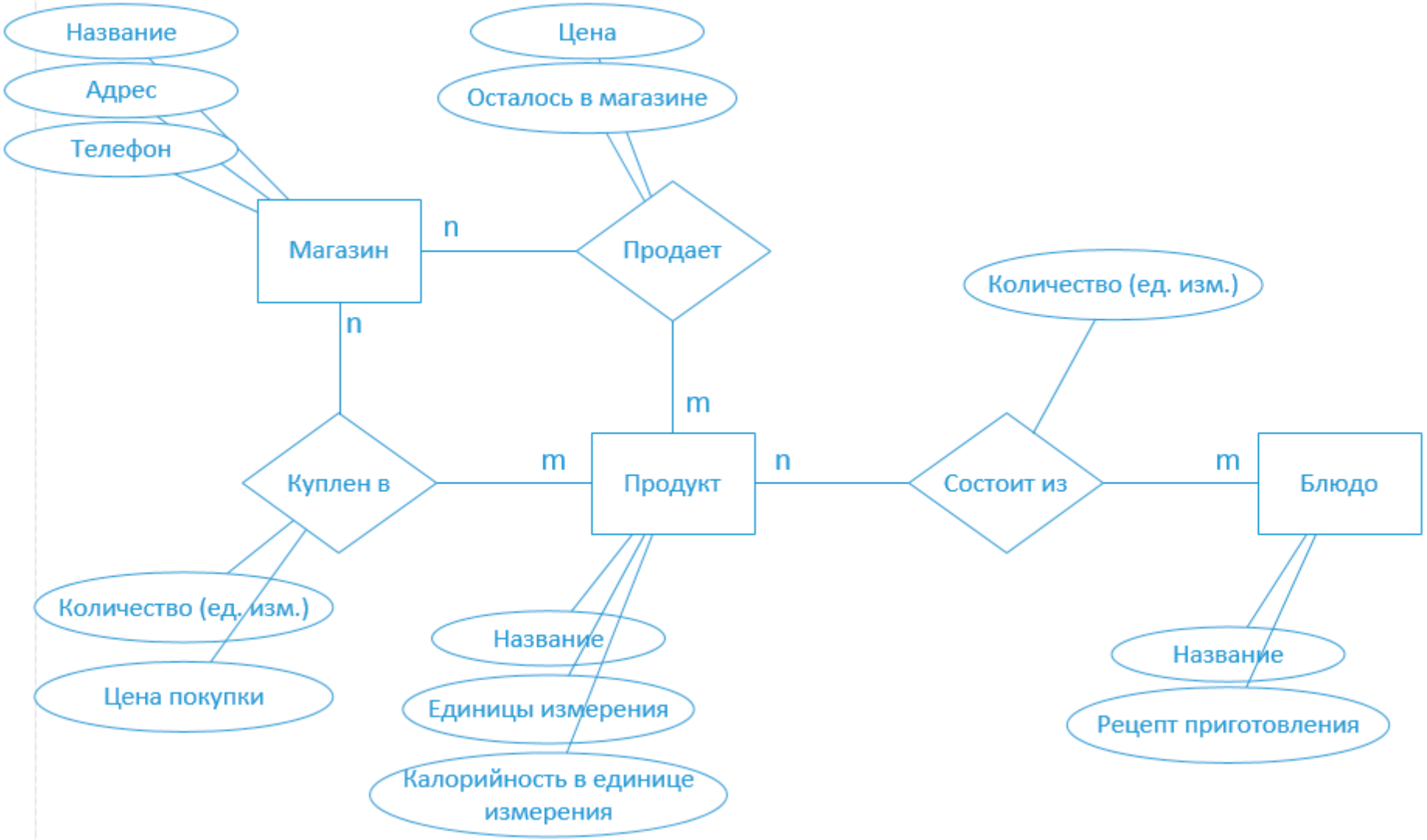
Проектирование предметной области

Далее выделяем атрибуты сущностей и атрибуты связей
В конце – добавляем ключи (идентификаторы), чтобы связи были через числа, а не через строки и т.п.

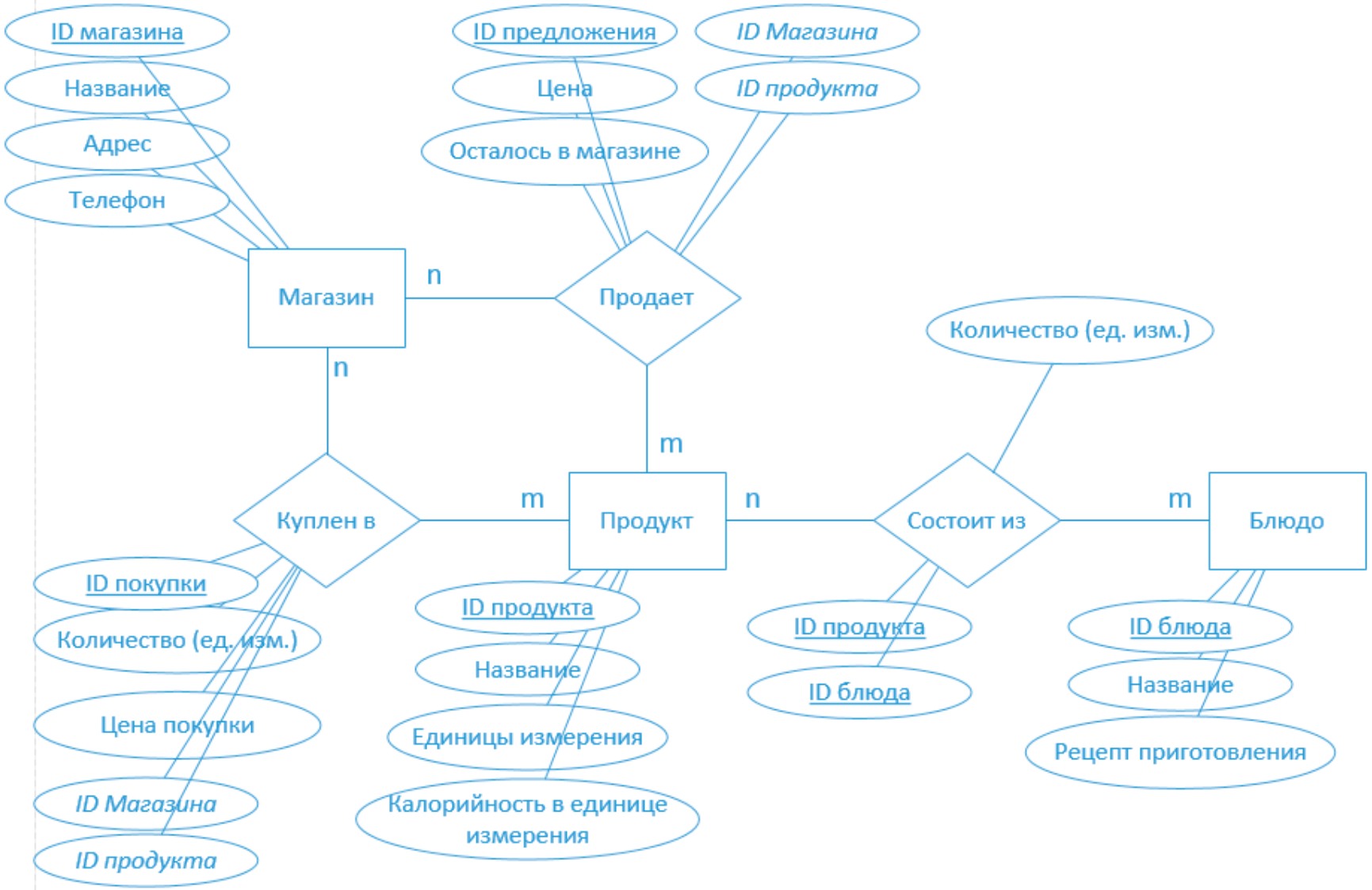
Проектирование предметной области



Проектирование предметной области



Проектирование предметной области



Программа к ЛР2

Программа, описывающая предметную область Кулинария