Московский Государственный Технический Университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Компьютерные системы и сети»

**Методическое пособие по лабораторной работе №2**

**«Создание прототипа ИС на базе MongoDB»**

Москва, 2019

*Цель работы*: научиться создавать БД в MongoDB.

# **Теоретическая часть**

## *Основные концепции MongoDB*

1. MongoDB — концептуально то же самое, что обычная, привычная нам база данных (или в терминологии Oracle — схема). Внутри MongoDB может быть ноль или более баз данных, каждая из которых является контейнером для коллекций.
2. База данных может иметь ноль или более «коллекций». Коллекция настолько похожа на традиционную реляционную «таблицу», что можно смело считать их одним и тем же.
3. Коллекции состоят из нуля или более «документов». Следуя логике реляционных БД, документ можно рассматривать как «строку».
4. Документы могут быть вложенными и это серьезное отличие от реляционных БД.
5. Документ состоит из одного или более «полей», которые подобны полям реляционных БД, правда в конкретном документе количество и типы полей могут варьироваться. В конечном счёте дело в том, что коллекция не содержит информации о структуре содержащихся в ней данных. Информацию о полях содержит каждый отдельный документ.
6. «Индексы» в MongoDB почти идентичны таковым в реляционных базах данных.
7. «Курсоры» в MongoDB заслуживают отдельного обсуждения. Важно понимать, что когда мы запрашиваем у MongoDB какие-либо данные, то она возвращает курсор, с которыми мы можем делать все что угодно — подсчитывать, пропускать определённое число предшествующих записей — при этом не загружая сами данные.

Подводя итог, MongoDB состоит из «баз данных», которые состоят из «коллекций». «Коллекции» состоят из «документов». Каждый «документ» состоит из «полей». «Коллекции» могут быть проиндексированы, что улучшает производительность выборки и сортировки. И наконец, получение данных из MongoDB сводится к получению «курсора», который отдаёт эти данные по мере надобности.

## *Работа с пользователями*

В MongoDB пользователи могут иметь привилегии в разных базах данных. При добавлении пользователя вы создаете пользователя в конкретной базе данных. Эта база данных является базой данных аутентификации для этого пользователя.

Для создания пользователя перейдите в БД, в которой нужно создать обычного пользователя: use my-database

Создайте пользователя *my-user* с правами readWrite на базу данных *my-database*:

db.createUser(

{

user: "my-user",

pwd: "passw0rd",

roles: [

{ role: "readWrite", db: "my-database" }

]

}

)

Также можно создать пользователя с различными правами на разные базы данных.

Для просмотра пользователей в текущей базе данных используются команды db.getUsers() или show users.

Для удаления пользователя из текущей базы данных используется команда db.dropUser("my-user").

## *Язык запросов MongoDB*

СУБД MongoDB относится к NoSQL базам данных, основной чертой которых является нереляционный характер и соответственно язык запросов, отличный от SQL. В MongoDB в качестве язык запросов используется JavaScript и JSON-структуры. Выбор столь нехарактерного языка запроса объясняется тем, что эта документо-ориентированная СУБД использует JSON-формат для представления документов и вывода результатов. Физически JSON-структуры хранятся в бинарном BSON формате.

#### Вставка нового документа

Внутри текущей базы данных создаются коллекции, вставка нового документа в коллекцию при помощи метода INSERT приводит к её автоматическому созданию. В качестве аргумента метод insert принимает JSON-объект, который служит телом документа. Ниже в базу данных вставляются документы с единственным ключом title, в качестве значения которого выступают названия различных СУБД.

db.mybase.insert({title: "MySQL"})

db.mybase.insert({title: "PostreSQL"})

db.mybase.insert({title: "MongoDB"})

db.mybase.insert([{title: "MS SQL"}, {title: "Oracle"}])

Формат JSON допускает создание, как единичных объектов, так и массивов.

Записи "MySQL", "PostgreSQL" и "MongoDB" вставляются отдельными вставками, в то время как "MS SQL" и "Oracle" при помощи массива, элементы которого в JSON заключаются в квадратные скобки. Для каждого документа автоматически создается идентификатор "\_id", который можно задавать вручную, с единственным условием, чтобы он оставался уникальным в рамках текущей базы данных. Идентификатор, который формируется по умолчанию, формируется по следующему алгоритму: в старших четырех байтах находится время создания записи в формате UNIXSTAMP, следующие три байта — идентификатор компьютера, следующие два — идентификатор процесса, последние три — локальный счетчик процесса.

Основная особенность MongoDB заключается в том, что документ или запись может быть настолько сложной по структуре, насколько допускает формат JSON. В качестве примера, в отдельной базе данных articles создадим набор статей, имеющий название и набор ключевых слов.

db.articles.insert({title: "Обзор NoSQL решений", tags: ["NoSQL", "MongoDB", "memcached", "CouchDB", "Riak", "HBase", "Redis"]})

db.articles.insert({title: "MongoDB", tags: ["NoSQL", "MongoDB"]})

db.articles.insert({title: "Redis", tags:["NoSQL","Redis"]})

Здесь помимо поля title, создается поле tags, представляющего собой массив ключевых слов. Элементы массивов и значения могут сами быть составными JSON-объектами и массивами. В результате одна запись в базе данных может представлять собой сложную структуру и значительно отличаться по своему составу от соседней (в отличие от традиционных реляционных баз данных, в которых структура записей в таблице строгая).

#### Типы данных

MongoDB поддерживает следующие типы данных:

* string — строки должны быть представлены в кодировке UTF-8;
* числа
  + - double — 8-байтные числа с плавающей точкой IEEE;
    - int — 4-байтное целое число;
    - long — 8-байтное целое число;
* datetime — календарный тип для хранения даты и времени, представляющий собой 8-байтное целое число, хранящее количество миллисекунд, прошедших с 1 января 1970 года;

При вставке чисел через JavaScript следует иметь в виду, что JavaScript поддерживает только числовой тип Number, соответствующий типу double.

Поэтому для явной вставки целых чисел типа int и long следует использовать классы NumberInt() и NumberLong(), соответственно.

db.numbers.save({num: NumberLong(5)});

#### Извлечение результатов

Для подсчета количества документов в текущей базе данных можно воспользоваться методом count(): db.mybase.count()

Для того, чтобы извлечь результаты, можно воспользоваться методом find():

**db.mybase.find()**

{"\_id": ObjectId("51eb8c2bb4d7d4d898b05fce"), "title": "MySQL"}

{"\_id": ObjectId("51eb905ab4d7d4d898b05fcf"), "title": "PostreSQL"}

{"\_id": ObjectId("51eb9061b4d7d4d898b05fd0"), "title": "MongoDB"}

{"\_id": ObjectId("51eb907db4d7d4d898b05fd1"), "title": "MS SQL"}

{"\_id": ObjectId("51eb907db4d7d4d898b05fd2"), "title": "Oracle"}

Методы допускают использование селекторов, например, для извлечения документа, соответствующего MySQL методу find() можно передать следующий JSON-объект

**db.mybase.find({"title" : "MySQL"})**

{"\_id": ObjectId ("51eb9bb1303d105141c7d74b"), "title": "MySQL"}

#### Объекты и переменные JavaScript

Консоль mongo позволяет не только задействовать предопределенные объекты, но и вводить свои. Ниже вводится JavaScript-объект where, в котором формируется условие поиска, далее в метод find() может передаваться объект.**var where = {}**

**where['title'] = "MySQL"**

**db.mybase.find(where)**

{"\_id": ObjectId("51eba53f303d105141c7d751"), "title": "MySQL"}

Консоль поддерживает и более сложные приемы JavaScript программирования, например, создание собственных методов.

#### Регулярные выражения

В качестве селектора могут выступать не только строки, но и регулярные выражения, например, для извлечения всех записей поле title, которые начинаются с символа M можно воспользоваться регулярным выражением /^M/ передав его методу find()**db.mybase.find({title: /^M/});**

{"\_id": ObjectId("51f4dae823d2a4ef32d25ec4"), "title": "MySQL"}

{"\_id": ObjectId("51f4dae823d2a4ef32d25ec6"),"title": "MongoDB"}

{"\_id": ObjectId("51f4dae923d2a4ef32d25ec7"),"title": "MS SQL"}

#### Сортировка результатов

Для сортировки результатов используется метод sort(), который принимает в качестве параметров JSON-структуру, ключом в которой выступает название сортируемого поля, а в качестве значения выступает целое число: положительное — прямая сортировка, отрицательная — обратная

**db.mybase.find({title: /^M/}).sort({title: 1});**

{"\_id": ObjectId("51f53e56a631742542dda011"), "title": "MS SQL"}

{"\_id": ObjectId("51f53e55a631742542dda010"), "title": "MongoDB"}

{"\_id": ObjectId("51f53e55a631742542dda00e"), "title": "MySQL"}

**db.mybase.find({title: /^M/}).sort({title: -1});**

{"\_id": ObjectId("51f53e55a631742542dda00e"), "title": "MySQL"}{"\_id": ObjectId("51f53e55a631742542dda010"), "title": "MongoDB"}

{"\_id": ObjectId("51f53e56a631742542dda011"), "title": "MS SQL"}

Сортировать можно по нескольким полям одновременно. Создадим таблицу с двумя полями fst и snd

**db.sortexmpl.insert({fst: 2, snd: 5})**

**db.sortexmpl.insert({fst: 1, snd: 12})**

**db.sortexmpl.insert({fst: 2, snd: 2})**

**db.sortexmpl.insert({fst: 1, snd: 20})**

**db.sortexmpl.insert({fst: 2, snd: 7})**

**db.sortexmpl.insert({fst: 1, snd: 6})**Ниже приводится пример сортировки по двум полям одновременно

**db.sortexmpl.find().sort({fst: 1, snd: 1})**

{"\_id": ObjectId("51f56e07a631742542dda01b"), "fst": 1, "snd": 6}

{"\_id": ObjectId("51f56e07a631742542dda017"), "fst": 1, "snd": 12}

{"\_id": ObjectId("51f56e07a631742542dda019"), "fst": 1, "snd": 20}

{"\_id": ObjectId("51f56e07a631742542dda018"), "fst": 2, "snd": 2}

{"\_id": ObjectId("51f56e07a631742542dda016"), "fst": 2, "snd": 5}

{"\_id": ObjectId("51f56e07a631742542dda01a"), "fst": 2, "snd": 7}

#### Ограничение выборки по полям документа

По умолчанию выборка содержит все поля документа, однако, в том случае, если требуется выбрать лишь конкретные поля, методам find() и findOne() можно передавать второй аргумент в виде JSON структуры, с ключами, совпадающими с названиями столбцов и значениями 1, если поле должно попадать в выборку и 0, если его необходимо исключить из выборки. В следующем запросе извлекаются только названия title, идентификатор \_id исключается из выборки:**db.mybase.find({title: /^M/}, {title: 1, \_id: 0}).sort({title: 1});**

{"title": "MS SQL"}

{"title": "MongoDB"}

{"title": "MySQL"}

Более того, не обязательно указывать включаемые поля, достаточно перечислить поля, которые не должны попасть в выборку

**db.mybase.find({title: /^M/}, {\_id: 0}).sort({title: 1});**

Если в условии нет необходимости, то в качестве первого запроса метода find() передается пустой селектор

**db.mybase.find({}, {\_id: 0}).sort({title: 1});**

{"title": "MS SQL"}

{"title": "MongoDB"}

{"title": "MySQL"}

{"title": "Oracle"}

{"title": "PostreSQL"}

#### Ограничение выборки по количеству документов

Для того чтобы извлечь лишь одно значение из полученной выборки, можно воспользоваться методом findOne()

**db.mybase.findOne({title: /^M/});**

{"\_id": ObjectId("51f4dae823d2a4ef32d25ec4"), "title": "MySQL"}

Однако, к методу findOne() прибегают чаще в тех ситуациях, когда ожидается, что результирующая коллекция будет содержать лишь один документ. В тех же случаях, когда следует ограничить выборку несколькими документами, рекомендуется использовать метод limit(), который принимает в качестве аргумента количество извлекаемых документов. Следующий запрос вернёт только 2 статьи из коллекции mybase.

**db.mybase.find({title: /^M/}).sort({title: 1}).limit(2);**

{"\_id": ObjectId("51f53e56a631742542dda011"), "title": "MS SQL"}

{"\_id": ObjectId("51f53e55a631742542dda010"), "title": "MongoDB"}

Для организации постраничной навигации может потребоваться метод skip(), который позволяет пропустить в выборке количество аргументов, указанных в его параметре. Так, чтобы извлечь содержимое следующей «страницы»:

**db.mybase.find({title: /^M/}).sort({title: 1}).skip(2).limit(2);**

{"\_id": ObjectId("51f53e55a631742542dda00e"), "title": "MySQL"}

При использовании больших значений в качестве параметра метода skip() следует помнить, методу приходится пропускать это значение записей, прежде чем добраться до извлекаемых. В этом случае разумно отказаться от skip() и использовать операторы сравнения

#### Логические операторы

JSON-структура в методе find(), может содержать несколько полей, при этом условия объединяются по логике AND (И). В следующем примере выбираются базы данных, начинающиеся с символа M, и использующие JavaScript в качестве языка запросов. Но сначала создадим коллекцию lang, содержащую документы, состоящие из двух полей, title — название базы данных и query — используемый ею язык запросов.

db.lang.insert({title: "MySQL", query: "SQL"})

db.lang.insert({title: "PostgreSQL", query: "SQL"})

db.lang.insert({title: "MongoDB", query: "JavaScript"})

**db.lang.find({query: "JavaScript", title: /^M/})**

{"\_id": ObjectId("51f55286a631742542dda015"), "title": "MongoDB", "query": "JavaScript"}

Для формирования логики OR (ИЛИ) предназначен оператор **$or**, который оперирует массивом аргументов как запрос, извлекающий записи или начинающиеся с символа M, или использующие в качестве языка запроса SQL может выглядеть следующим образом

**db.lang.find({$or: [{query: "SQL"}, {title: /^M/}]})**

{"\_id": ObjectId("51f55285a631742542dda013"), "title": "MySQL", "query": "SQL"}

{"\_id": ObjectId("51f55285a631742542dda014"), "title": "PostgreSQL", "query": "SQL"}

{"\_id": ObjectId("51f55286a631742542dda015"), "title": "MongoDB", "query": "JavaScript"}

В таблице 1. представлены операторы сравнения и логические операторы MongoDB. Операторы можно использовать как по отдельности, так и в комбинации

Таблица 1 – операторы MongoDB

|  |  |  |
| --- | --- | --- |
| **Оператор SQL** | **MongoDB** | **Описание** |
|  |  | Меньше |
|  |  | Меньше или равно |
|  |  | Больше |
|  |  | Больше или равно |
|  |  | Не равно |
|  |  | Отрицание |
|  |  | Проверка существования поля |
|  |  | Или |
|  |  | Не или |
|  |  | Соответствие регулярному выражению |
|  |  | Соответствие всех полей вложенного документа |
|  |  | Соответствие размеру массива |
|  |  | Соответствие, если поле имеет указанный тип |

**db.lang.find({\_id: {$gt: ObjectId("51f55285a631742542dda013")}})**

{"\_id": ObjectId("51f55285a631742542dda014"), "title": "PostgreSQL", "query" : "SQL" }

{"\_id": ObjectId("51f55286a631742542dda015"), "title": "MongoDB", "query": "JavaScript"}

**db.lang.find({\_id: {$lt: ObjectId("51f55285a631742542dda015")}})**

{"\_id": ObjectId("51f55285a631742542dda013"), "title": "MySQL", "query": "SQL"}

{"\_id": ObjectId("51f55285a631742542dda014"), "title": "PostgreSQL", "query": "SQL"}

**db.lang.find({\_id: {$gt: ObjectId("51f55285a631742542dda013"), $lt: ObjectId("51f55285a631742542dda015")}})**

{"\_id": ObjectId("51f55285a631742542dda014"), "title": "PostgreSQL", "query": "SQL"}

В оболочке mongo можно составлять более сложные программы. Так, например, двойное условие из предыдущего примера можно оформить в виде объекта, подставляемого в JSON-структуру

**var conditions = {}**

**conditions['$gt'] = ObjectId("51f55285a631742542dda013")**

**ObjectId("51f88c41b49ab634fd78bf93")**

**conditions['$lt'] = ObjectId("51f55285a631742542dda015")**

**ObjectId("51f88c41b49ab634fd78bf95")**

**db.lang.find(conditions)**

**db.lang.find({\_id: conditions})**

{"\_id": ObjectId("51f55285a631742542dda014"), "title": "PostgreSQL", "query": "SQL"}

#### Работа со списками

В таблице 2 показаны операторы, предназначенные для работы со списками.

Таблица 2 – операторы для работы со списками

|  |  |  |
| --- | --- | --- |
| **Оператор SQL** | **MongoDB** | **Описание** |
|  |  | Входит в список |
|  |  | Не входит в список |
|  |  | Одновременное совпадение набора элементов |

Оператор $in извлекает записи, для которых заданное значение совпадает хотя бы с одним из списка. Ниже выводится запрос, извлекающий записи, поле title которых совпадает с одним из значений в списке.

['MongoDB', 'MySQL']:

**db.lang.find({title: {$in: ['MongoDB', 'MySQL']}})**

{ "\_id" : ObjectId("51f55285a631742542dda013"), "title" :

"MySQL", "query" : "SQL" }

{ "\_id" : ObjectId("51f55286a631742542dda015"), "title" :

"MongoDB", "query" : "JavaScript" }

Для извлечения всех записей, не входящих в список ['MongoDB', 'MySQL'] можно воспользоваться оператором $nin

**db.lang.find({title: {$nin: ['MongoDB', 'MySQL']}})**

{ "\_id" : ObjectId("51f88c41b49ab634fd78bf94"), "title" :

"PostgreSQL", "query" : "SQL" }

#### Обновление документов

Для обновления используется метод update, первый аргумент которого определяет список обновляемых документов, второй — как отобранные документы модифицируются. Следующая команда добавляет в документ с названием "MongoDB" дополнительное поле nosql, со значением "true":

**db.mybase.update({"title": "MongoDB"}, {$set: {nosql: "true"}})**

{"\_id": ObjectId("51eb9bb1303d105141c7d74d"), "nosql": "true", "title": "MongoDB"}

Если обновлению подвергается только одно поле, следует обязательно использовать оператор $set. Если его опустить, вместо обновления поля, будет обновлен весь документ:

**db.mybase.update({"title": "MongoDB"}, {nosql: "true"})**

{"\_id": ObjectId("51f9582829b045085952815f"), "nosql": "true"}

Для удаления дополнительного поля nosql вместо оператора $set следует подставить $unset

**db.mybase.update({"title": "MongoDB"}, {$unset: {nosql: "none"}})**

**db.mybase.find({"title": "MongoDB"})**

{"\_id": ObjectId("51eb9bb1303d105141c7d74d"), "title": "MongoDB"}

Для того чтобы добавить ключевое слово в массив тэгов tags базы данных articles следует воспользоваться оператором $addToSet

**db.articles.update({tags: "Redis"}, {$addToSet: {tags: "Обзор"}})**

Если запросить состав текущей базы данных при помощи метода find(), то можно заметить, что обновился только один документ из двух, которые имеют ключевое слово "Redis". Для того, чтобы обновить все документы, следует передать четвертому параметру метода update() значение true, что заставит MongoDB обновить все найденные документы

db.articles.update({tags: "Redis"}, {$addToSet: {tags: "Обзор"}}, false, true)

Ниже приводится сводная таблица 3 по операторам, используемым совместно с методом update().

Таблица 3 – Операторы, используемые совместно с методом update()

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
|  | обновление или создание поля |
|  | удаление поля |
|  | увеличение значения поля на заданное число |
|  | удаляет последний (или первый) элемент массива |
|  | помещает несколько элементов в массив |
|  | помещает новый элемент в массив |
|  | помещает новый элемент в массив (исключаются дубликаты) |
|  | удаляет из массива значение (при его наличии) |
|  | удаляет из массива все подходящие значения |

#### Удаление документов

Полностью удалить документы из текущей базы данных можно при помощи метода remove().

**db.mybase.remove()**

Точно также, как и другие методы, в качестве параметра метод может принимать селекторы, например, удалить все записи с ключом nosql равным "true" можно при помощи следующей команды

**db.mybase.remove({nosql: "true"})**

## *Использование MongoDB в Python*

С MongoDB можно работать не только через командную строку, но и при помощи языков программирования. Например, для использования MongoDB в приложениях на Python нужно подключить пакет *pymongo*. Ниже приведены основные команды для подключения к БД, создание коллекций, документов и т. д.:

# импорт (подключение пакета) pymongo

import pymongo

# соединение с сервером базы данных

# (по умолчанию подключение осуществляется на localhost:27017)

conn = pymongo.Connection()

# подключение к другому серверу, на другой порт

conn = pymongo.Connection('localhost', 27017)

# выбор базы данных mydb

db = conn.mydb

# БД можно выбрать и так

db = conn['mydb']

# выбор коллекции документов

coll = db.mycoll

# альтернативный выбор коллекции документов

coll = db['mycoll']

# добавление документа в коллекцию,который содержит поля name и

# surname - имя и фамилия

doc = {"name":"Иван", "surname":"Иванов"}

coll.save(doc)

# альтернативное добавление документа

coll.save({"name":"Петр", "surname":"Петров"})

# вывод всех документов из коллекции coll

for men in coll.find():

print men

# вывод фамилии людей с именем Петр

for men in coll.find({"name": "Петр"})

print men["surname"]

# подсчет количества людей с именем Петр

print coll.find({"name": "Петр"}).count()

# добавление ко всем документам нового поля sex - пол

coll.update({}, {"$set":{"sex": "мужской"}})

# всем Петрам делаем фамилию Новосельцев и возраст 25 лет

coll.update({"name": "Петр"}, {"surname": "Новосельцев", "age": 25})

# увеличиваем всем Петрам возраст на 5 лет

coll.update({"name": "Петр"}, {"$inc": {"age": 5}})

# сбрасываем у всех документов поле name

coll.update({}, {"$unset": {"name": 1}})

# удаляем людей с возрастом более 20 лет

# другие условия $gt - больше, $lt - меньше,

# $lte - меньше или равно, $gte - больше или равно, $ne - не равно

coll.remove({"age": {"$gt": 20}})

# удаление всех документов коллекции

coll.remove({})

# **Практическая часть**

В данной лабораторной работе Вам предстоит разработать фрагмент базы данных системы, предметную область которой кратко описана в Приложении 1. Конкретную предметную область можно получить, взяв вариант совпадающий с вашим порядковым номером в списке группы. Надо сказать, что приведенные схемы весьма общи, а иногда и не очень корректны, поэтому Вы можете изменить их по вашему усмотрению, оставаясь, конечно, в рамках предметной области. Для вашей предметной области необходимо выделить минимум 2 бизнес-процесса, которые поддаются автоматизации (это не значит, что вы должны их автоматизировать). По этим бизнес-процессам надо **выделить сущности предметной области**, которые создают костяк информационной модели предметной области. Далее нужно **прорисовать** примерную **схему, показывающую связи** **сущностей**. Не надо прорабатывать подробную схему таблиц как при работе с реляционной БД, да и можно ли говорить о «схеме» в NoSQL СУБД… Вам предлагается **создать интерфейсы к системе учета данных** вашей предметной области **с минимальным функционалом**:

* ввод и сохранение данных о базовых сущностях (можно просто скрипт создающий базу данных и заполняющую ее тестовыми значениями);
* поиск данных о базовых сущностях в соответствии с построенными бизнес-процессами (необходимо разработать интерфейс для поиска сущностей и скрипты выполняющие этот поиск);
* вывод и отображение данных хранящихся в БД.

Данные должны храниться в MongoDB, а инструментарий по построению интерфейса выбирается на ваше усмотрение.

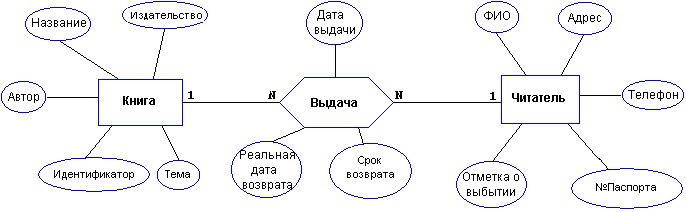
Отчёт по лабораторной работе должен включать:

* схемы бизнес процессов;
* схему взаимодействия сущностей;
* тексты скриптов для создания и заполнения коллекций;
* тексты скриптов для поиска и выдачи данных по базовым сущностям;
* скриншоты интерфейсов и заполненных коллекций.

Приложение1. Схемы предметных областей

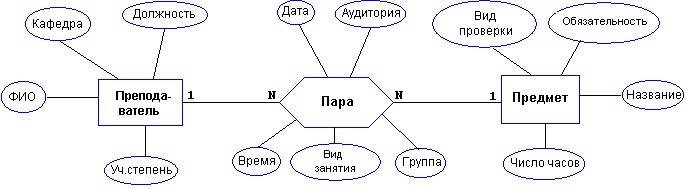
# Вариант №1

Предметная область для практических заданий: **Библиотека**



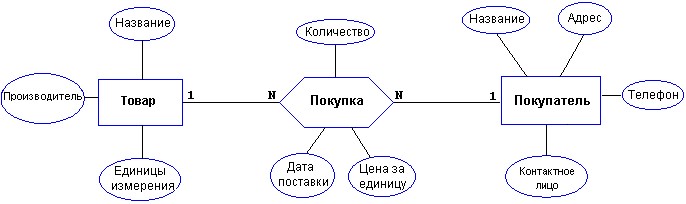
# Вариант №2

Предметная область для практических заданий: **Университет**



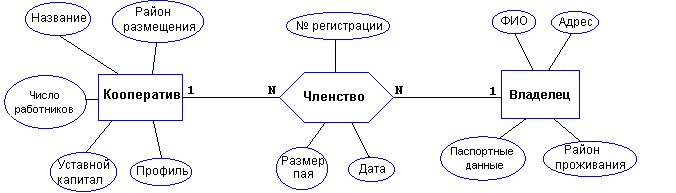
# Вариант №3

Предметная область для практических заданий: **Отдел продаж**



# Вариант №4

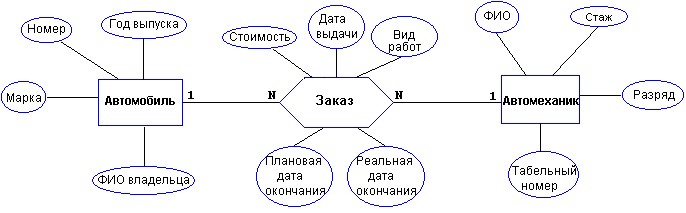
Предметная область для практических заданий: **Кооператив**



Примечание: профиль - продуктовый, галантерейный, канцелярский и т.п.

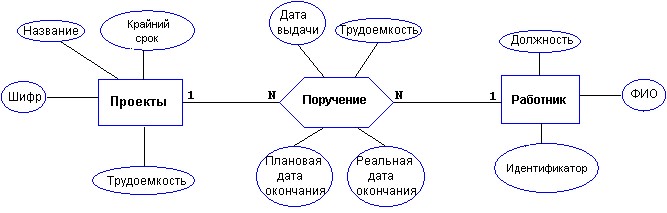
# Вариант №5

Предметная область для практических заданий: **Автомастерская**



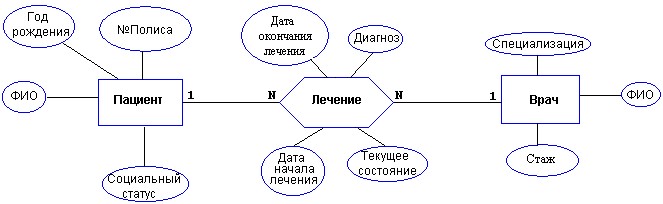
# Вариант №6

Предметная область для практических заданий: **Управление проектом**



# Вариант №7

Предметная область для практических заданий: **Поликлиника**

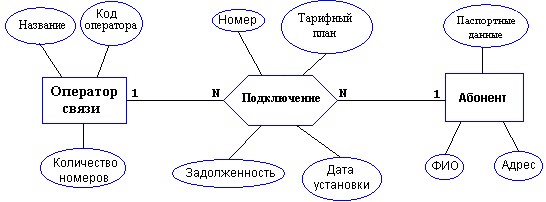


Текущее состояние - средней тяжести, тяжелое, направлен в стационар, умер. Социальный статус пациента - учащийся, работающий, врем. неработающий, инвалид, пенсионер

Специализация врача - терапевт, хирург и т.п.

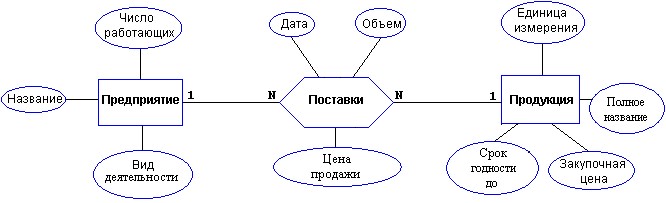
# Вариант №8

Предметная область для практических заданий: **Сотовая связь**



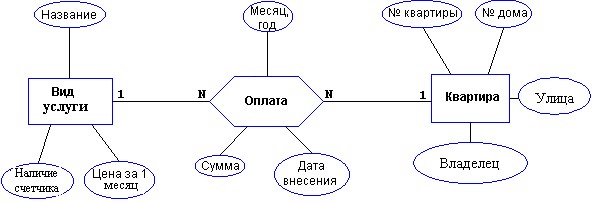
# Вариант №9

Предметная область для практических заданий: **Поставки**



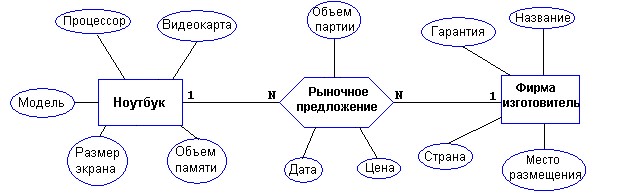
# Вариант №10

Предметная область для практических заданий: **Домоуправление**



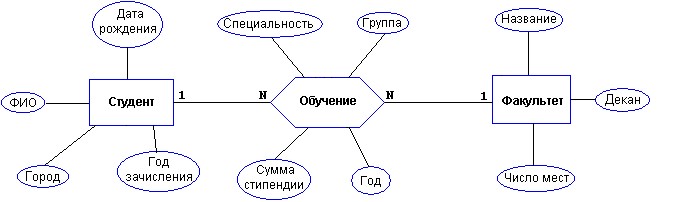
# Вариант №11

Предметная область для практических заданий: **Ноутбуки**



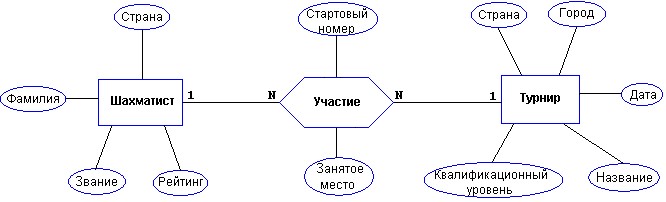
# Вариант №12

Предметная область для практических заданий: **Деканат**



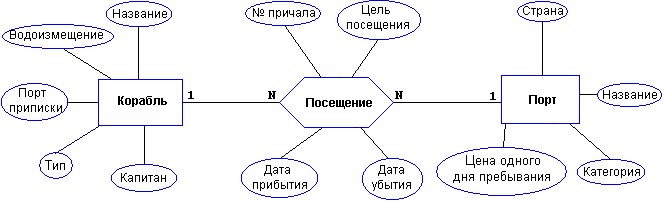
# Вариант №13

Предметная область для практических заданий: **Шахматы**



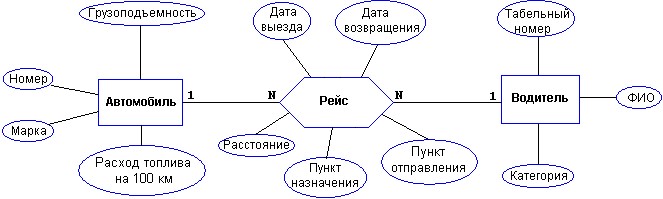
# Вариант №14

Предметная область для практических заданий: **Судоходство**



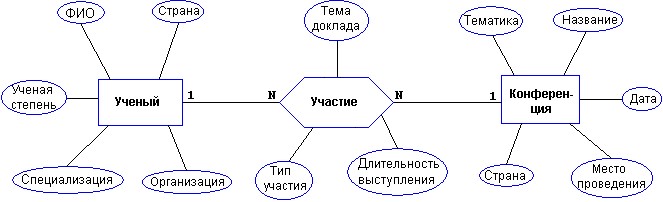
# Вариант №15

Предметная область для практических заданий: **Грузоперевозки**



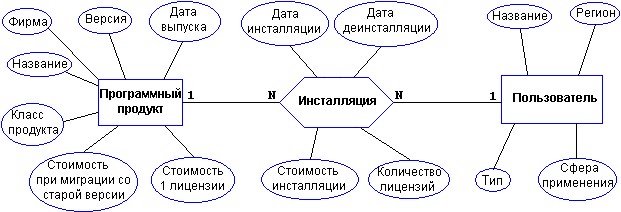
# Вариант №16

Предметная область для практических заданий: **Научные конференции**



# Вариант №17

Предметная область для практических заданий: **Программные продукты**

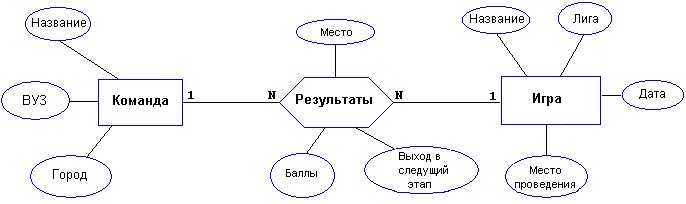


Класс: ОС, сервер приложений, СУБД, Web-сервер, система программирования и т.д.

Тип пользователя: индивидуальный, корпоративный, совместный, групповой и др.

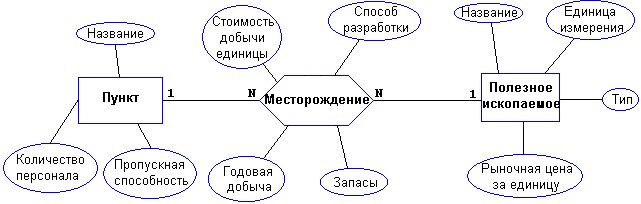
# Вариант №18

Предметная область для практических заданий: **КВН**



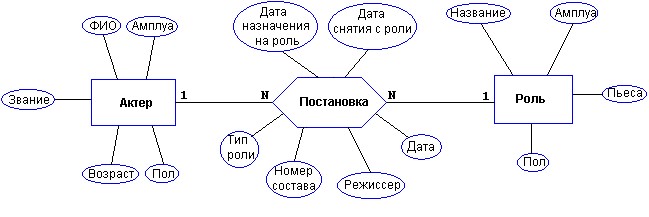
# Вариант №19

Предметная область для практических заданий: **Добыча ресурсов**



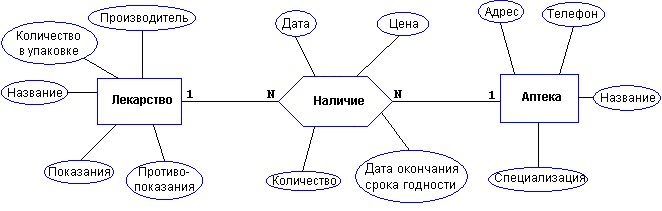
# Вариант №20

Предметная область для практических заданий: **Театр**



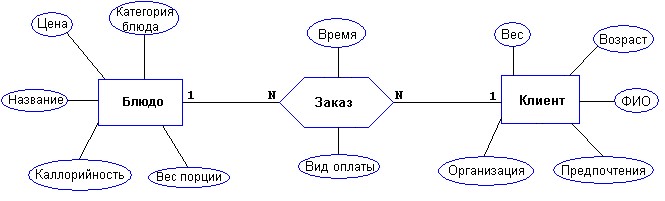
# Вариант №21

Предметная область для практических заданий: **Справочная аптек**



# Вариант №22

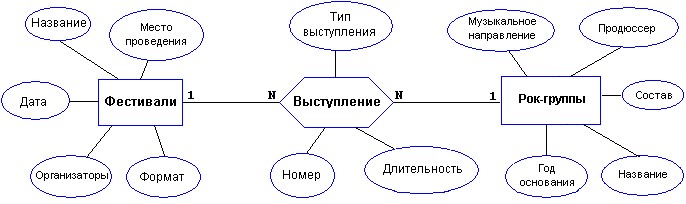
Предметная область для практических заданий: **Столовая**



Категория блюда: первое, второе, гарнир, десерт и т.д.

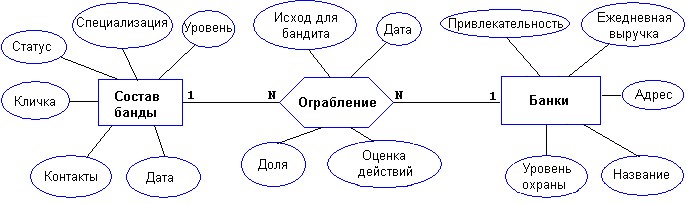
# Вариант №23

Предметная область для практических заданий: **Рок-группы**



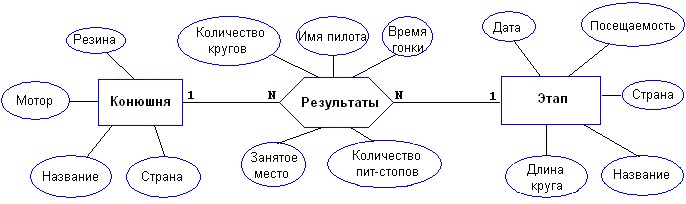
# Вариант №24

Предметная область для практических заданий: **ОПГ**



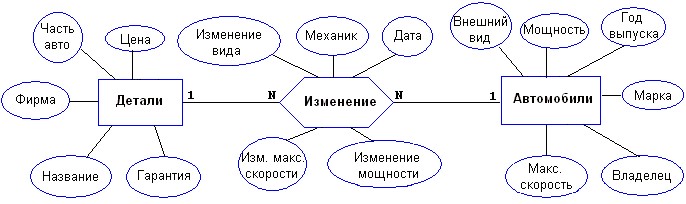
# Вариант №25

Предметная область для практических заданий: **Формула 1**



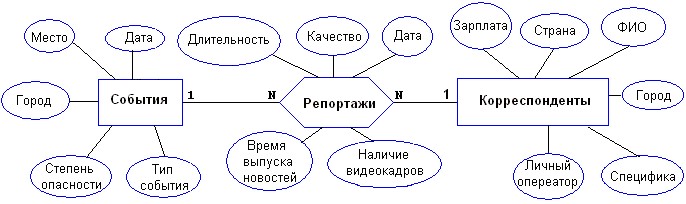
# Вариант №26

Предметная область для практических заданий: **Тюнинг**



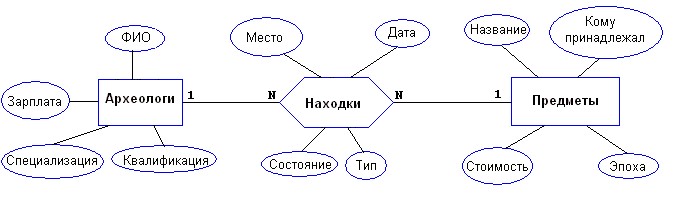
# Вариант №27

Предметная область для практических заданий: **Тележурналистика**



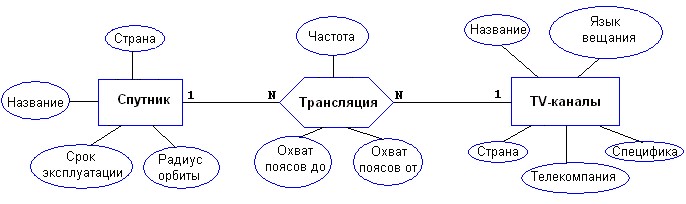
# Вариант №28

Предметная область для практических заданий: **Археология**



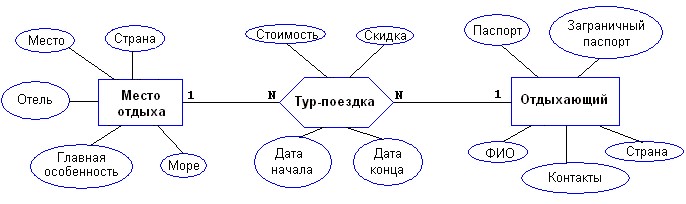
# Вариант №29

Предметная область для практических заданий: **Телевещание**

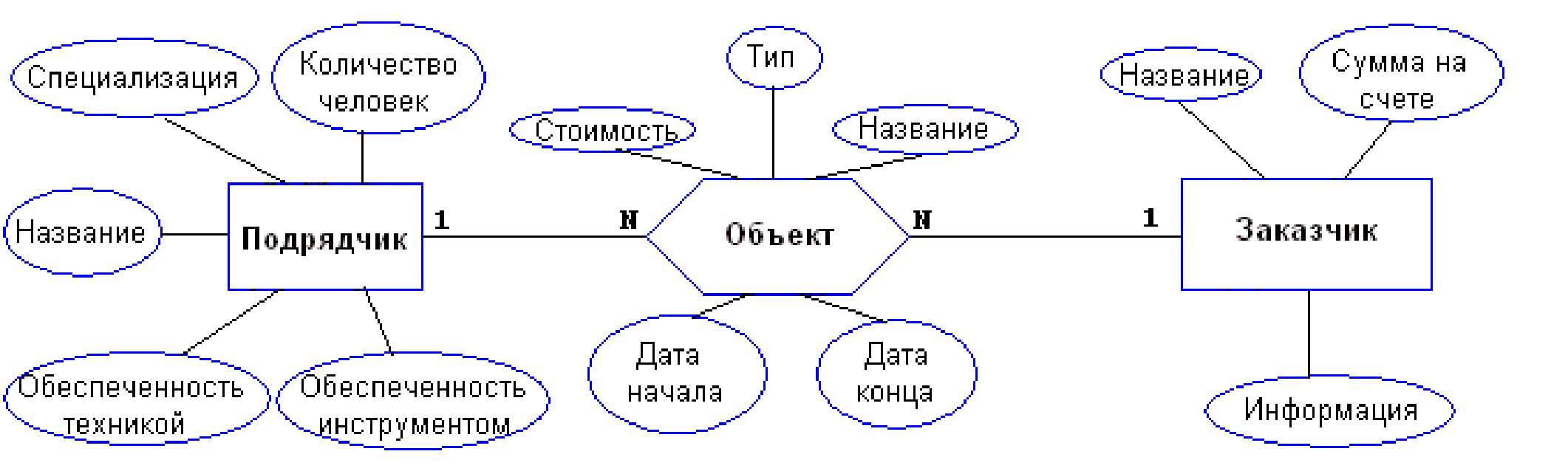


# Вариант №30

Предметная область для практических заданий: **Турфирма**



# Вариант№31

Предметная область для практических заданий: **Строительство**