Министерство науки и образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени

Н.Э. Баумана

(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)



**УЧЕБНО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО**

**КУРСУ «БАЗЫ ДАННЫХ»**

**Лабораторная работа №4**

**«Основы MongoDB»**

Авторы:

Кудрявцев А.П., [kudryavtsevap@bmstu.ru](mailto:kudryavtsevap@bmstu.ru)

Фомин М.М.

Москва, 2025

1. База данных

Если в реляционных БД содержимое составляют таблицы, то в MongoDB база данных состоит из коллекций. Каждая коллекция имеет свое уникальное имя - произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания.

В отличие от реляционных баз данных MongoDB не использует табличное устройство с четко заданным количеством столбцов и типов данных. MongoDB является документо-ориентированной системой, в которой центральным понятием является документ.

Документ можно представить как объект, хранящий некоторую информацию. В некотором смысле он подобен строкам в реляционных СУДБ, где строки хранят информацию об отдельном элементе. Например, типичный документ:

{

surname: "Иванов",

name: "Иван",

fname: "Иванович",

bdate: "2005.06.26",

group: {

name: "ИУ6",

year: 2023

}

}

Документ представляет набор пар ключ-значение. Например, в выражении surname: "Иванов" surname представляет ключ, а "Иванов» - значение.

Ключи представляют строки. Значения же могут различаться по типу данных. В данном случае у нас почти все значения также представляют строковый тип, и лишь один ключ group ссылается на отдельный объект.

2. Типы данных

Наиболее распространенные типы:

**Null** – нулевой тип используется для обозначения неопределенного значения:

{"x": null}

**Логический тип** – хранящий логические значения true или false:

{"x": true}

**Число** – по умолчанию оболочка использует 64-битные числа с плавающей

точкой. Таким образом, в оболочке эти числа выглядят «нормально»:

{"x": 3.14}

{"x": 3}

В случае с целыми числами используйте классы NumberInt или NumberLong, которые обозначают 4-байтовые или 8-байтовые целые числа со знаком соответственно:

{"x": NumberInt ("3")}

{"x": NumberLong ("3")}

**Строка** – любая строка символов в кодировке UTF-8 может быть представлена

с использованием типа строки:

{"x": "Московский Государственный Технический Университет"}

**Дата** – MongoDB хранит даты в виде 64-битных целых чисел, обозначающих миллисекунды с момента эпохи Unix (1 января 1970 г.). Часовой пояс не сохраняется:

{"x": new Date()}

**Регулярное выражение** – запросы могут использовать регулярные выражения, используя синтаксис регулярных выражений JavaScript:

{"x": / foobar / i}

**Массив** – наборы или списки значений могут быть представлены в виде массивов:

{"x": ["a", "b", "c"]}

**Встраиваемый документ** – документы могут содержать целые документы, встроенные в качестве значений в родительский документ.

{

surname: "Иванов",

name: "Иван",

fname: "Иванович",

bdate: "2005.06.26",

group: {

name: "ИУ6",

year: 2023

}

}

В приведенном примере можно увидеть, что вложенные документы могут изменить способ работы с данными. В реляционной СУБД предыдущий

документ, вероятно, будет смоделирован как две отдельные строки в двух

разных таблицах (students и groups). В MongoDB мы можем встраивать документ group непосредственно в документ students. Таким образом, при

правильном использовании вложенные документы могут обеспечить более естественное представление информации.

**Идентификатор объекта** – это 12-байтовый идентификатор для документов:

{"x": ObjectId ()}

Каждый документ, хранящийся в MongoDB, должен иметь ключ \_id. Значение ключа \_id может быть любого типа, но по умолчанию используется ObjectId. В одной коллекции каждый документ должен иметь уникальное значение \_id, что гарантирует уникальную идентификацию каждого документа в коллекции.

ObjectId является типом по умолчанию для \_id. Если при вставке документа ключ \_id отсутствует, он будет добавлен автоматически во вставленный документ. И, если мы попробуем добавить в коллекцию два документа с одинаковым идентификатором, то добавится только один из них, а при добавлении второго мы получим ошибку.

**Двоичные данные** – это строка из произвольных байтов. Ими нельзя манипулировать из оболочки. Двоичные данные – единственный способ сохранять строки не в формате UTF-8 в базе данных.

**Код** – MongoDB также позволяет хранить произвольный код JavaScript в

запросах и документах:

{"x": function() { /\* ... \*/ }}

**Date** – хранить значения даты и времени. это 64-битное целое число со знаком, представляющее количество миллисекунд с эпохи Unix (1 января 1970 г.). Положительные числа представляют время, прошедшее с эпохи, а отрицательные числа представляют время, движущееся назад от эпохи. Поскольку тип даты не хранит дополнительную информацию, такую ​​как часовые пояса, это значение должен храниться отдельно, если он важен.

Создать новый объект можно Date двумя способами:

* new Date(): возвращает дату и время как Date объект.
* ISODate(): Возвращает дату и время как Date объект в формате ISO8601.

Оба метода new Date() и ISODate() создают Date объект, который упакован во ISODate() вспомогательную функцию.

Кроме того, вызов Date() функции без new конструктора возвращает дату и время в виде строки, а не Date объекта

**Timestamp** – применяется для хранения времени, которое используется в основном для внутренних целей.

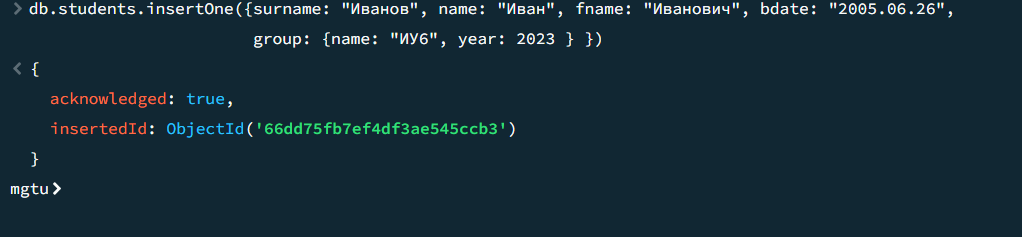
3. Добавление данных

Имя коллекции – произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания. В то же время имя коллекции не должно начинаться с префикса system., так как он зарезервирован для внутренних коллекций (например, коллекция system.users содержит всех пользователей базы данных). И также имя не должно содержать знака доллара – $.

Для добавления в коллекцию могут использоваться два ее метода:

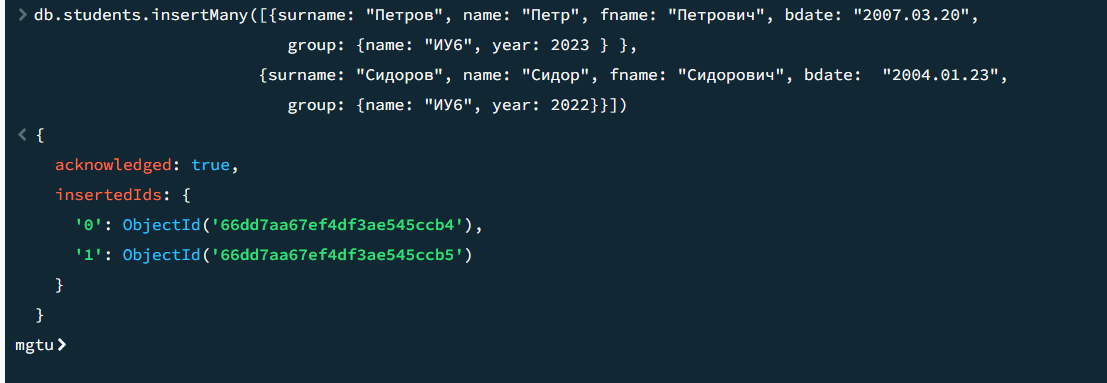
* insertOne() – добавляет один документ,
* insertMany() – добавляет несколько документов.

Используя GUI Compass, удалим коллекцию students, если она была создана и добавим один документ:



Если вам нужно вставить несколько документов в коллекцию, можно использовать метод insertMany. Этот метод позволяет передавать массив документов в базу данных, что гораздо более эффективно, потому что ваш код не будет бегать в базу данных и обратно ради каждого документа, а вставит их все сразу. Отправка десятков, сотен или даже тысяч документов за один раз может значительно ускорить процесс вставки.

И так, вставим два документа:



Хочу обратить ваше внимание на то, что вставка нескольких документов должна быть оформлена, как массив, т.е. документы должны быть заключены в квадратные скобки и отделяться запятыми.

При выполнении массовой вставки с использованием метода insertMany, если документ посреди массива выдает какую-либо ошибку, тогда то, что произойдет, зависит от того, выбрали ли вы упорядоченные или неупорядоченные операции. В качестве второго параметра insertMany можно указать документ опций. Укажите значение true для ключа ordered в документе параметров, чтобы обеспечить вставку документов в том порядке, в котором они были предоставлены. Укажите значение false, и MongoDB может изменить порядок вставок для повышения производительности. Упорядоченные вставки используются по умолчанию, если порядок не указан. В случае с упорядоченными вставками массив, переданный в insertMany, определяет порядок вставки. Если документ вызывает ошибку вставки, ни один документ за пределами этой точки в массиве не будет вставлен. В случае с неупорядоченными вставками MongoDB попытается вставить все документы независимо от того, приводят ли некоторые вставки к ошибкам. Приведем пример неупорядоченной вставки некоторого массива документов:

insertMany([{…}, {…}, {…}], { ordered: false })

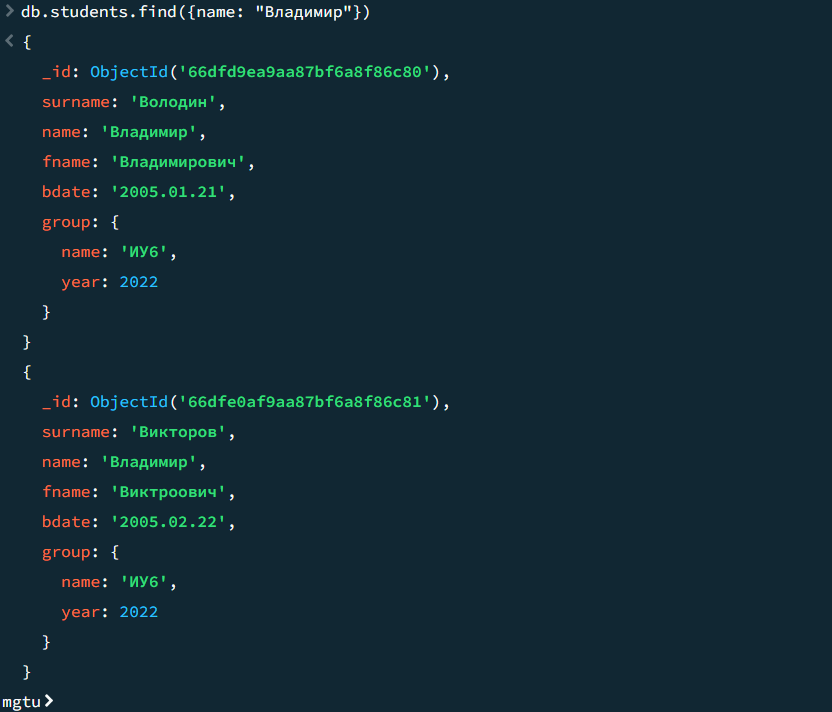
4. Извлечение данных

Наиболее простой способом выборки документов из коллекции представляет использование функции find(). Действие этой функции во многом аналогично обычному запросу SELECT \* FROM Table, который применяется в SQL и который извлекает все строки. Например, чтобы извлечь все документы из коллекции students, мы можем использовать команду:



4.1. Фильтрация данных

Рассмотрим случай, когда нам надо получить не все документы, а только те, которые удовлетворяют определенному требованию. Выведем документы с   
name = Владимир:

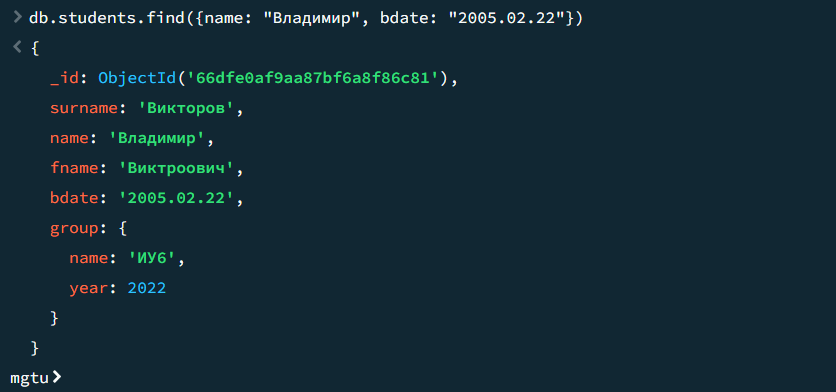


Такой запрос вывел нам два документа, в которых name=Владимир.

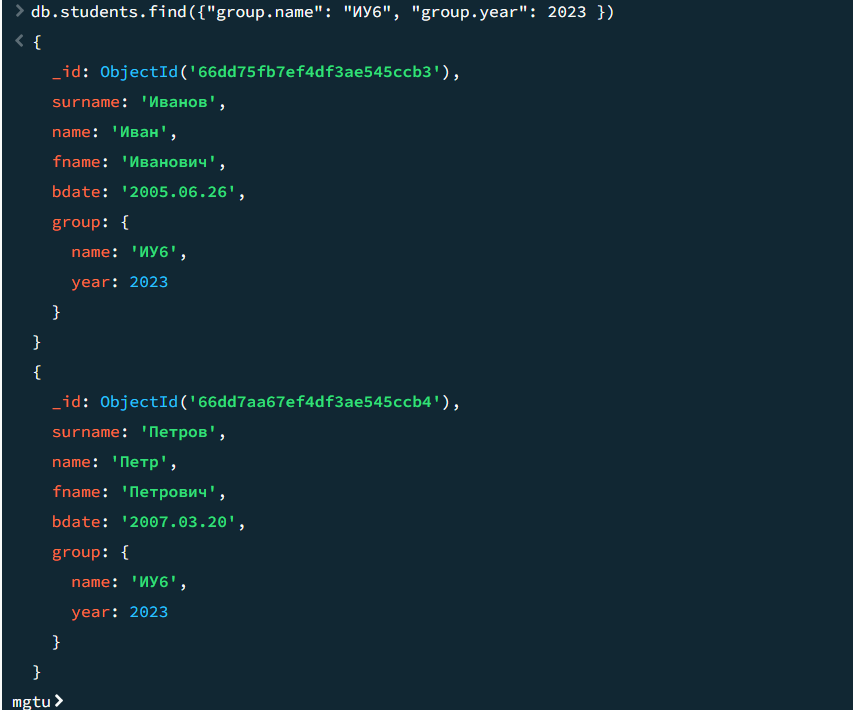
Теперь более сложный запрос: нам надо вывести те объекты, у которых name= Владимир и одновременно дата рождения bdate=2005.02.22. То есть на языке SQL это могло бы выглядеть так:

SELECT \* FROM Table WHERE Name='Владимир' AND bdate='2005.02.22'

Тогда мы можем написать следующий запрос:



Допустим, мы хотим вывести список студентов, обучающихся в одной группе. В этом случае необходимо выполнить запрос к вложенным документам. Найдем студентов обучающихся в группе ИУ6, поступивших в 2023 году:



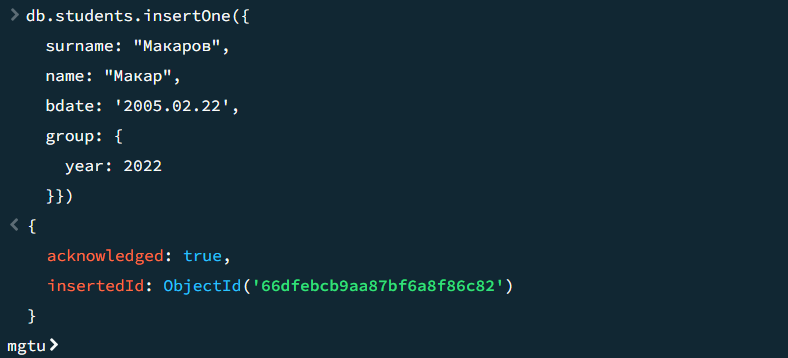
4.2. Поиск одиночного документа

Если все документы извлекаются функцией find, то одиночный документ извлекается функцией findOne:

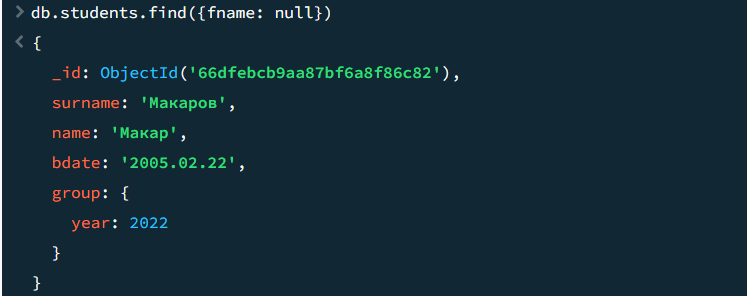


4.3. Фильтрация по отсутствующим свойствам

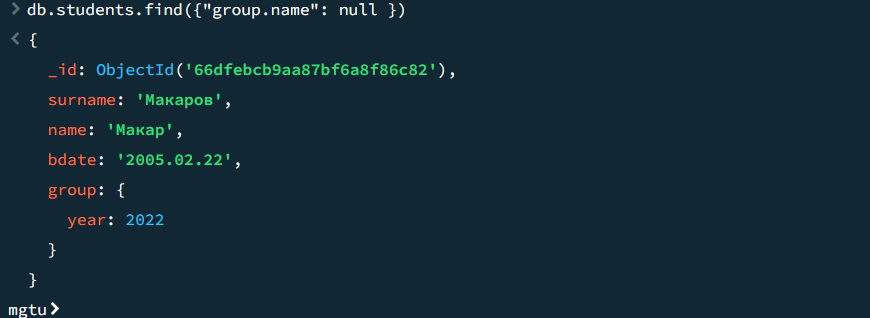
Добавим еще один документ, который не содержит атрибуты fname group.name (название кафедры):



Как видим документ успешно добавился в коллекцию. Что делать, если мы хотим получить подобные документы, в которых отсутствует определенное атрибуты, чтобы их исправить? В этом случае для атрибута передается значение null. Например, найдем все документы, где отсутствует fname:

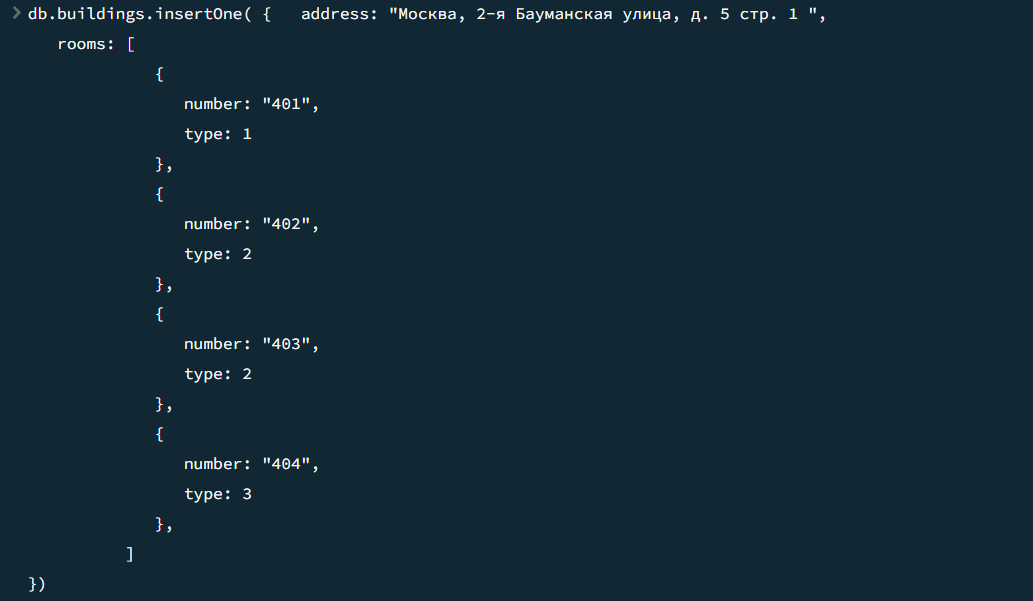


Теперь найдем документ без названия кафедры:

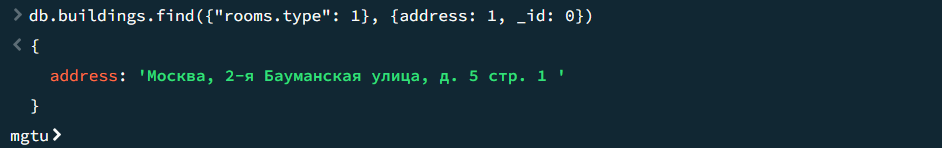


4.4. Фильтрация по элементам массива

Создадим коллекцию buildings (здания), где каждый документ кроме адреса будет содержать массив помещений rooms. Каждое помещение будет содержать атрибуты: номер помещения и назначение помещения (1 – для лекций, 2 – семинаров, 3 – лабораторных работ, 4 – прочее):



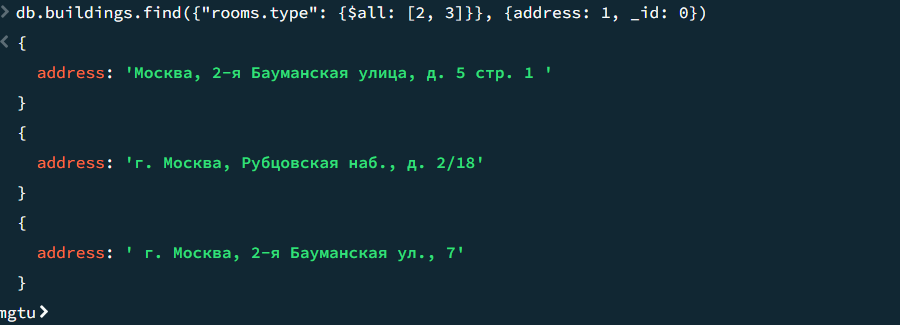
Добавим еще два документа в коллекцию. Найдем здания, в которых есть аудитории для чтения лекций:



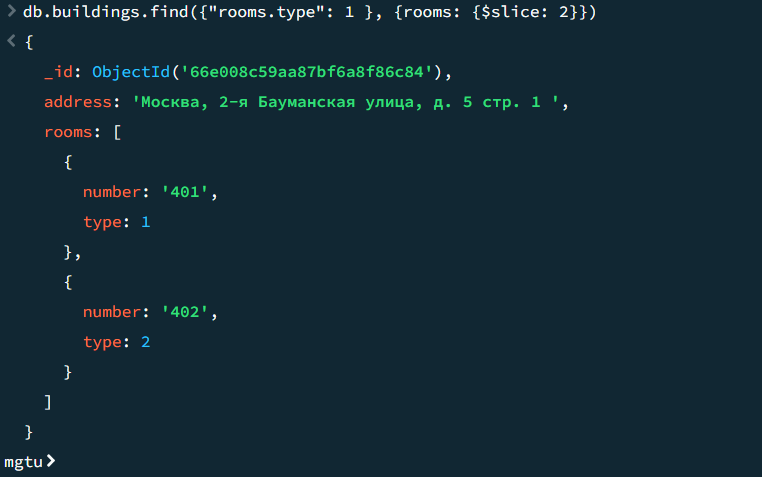
В этом запросе была добавлена опция {address: 1, \_id: 0}, чтобы вывести только адрес здания (о выводе только необходимых полей будет рассказано чуть ниже).

Если вам нужно сопоставить массивы по нескольким элементам, можно

использовать оператор "$all". Найдем те здания, в которых можно проводить семинары и лабораторные работы:



Для того, чтобы выводить подмножество элементов для ключа массива, служит оператор $slice. Например, выведем список первых двух помещений для какого-то корпуса:



Если вместо {$slice: 2} напишем {$slice: -2}, то выведутся два последних элемента массива.

Если вы хотите запросить определенный элемент массива, можно указать индекс, используя синтаксис ключ.индекс. В массивах индексация всегда начинается с 0, если необходимо получить первый элемент массива, то пишем rooms.0. Выведем все документы, у которых первый элемент имеет тип равный 1:



4.5. Указываем, какие ключи нужно вернуть (проекция)

Документ может иметь множество полей, но не все эти поля нам могут быть нужны и важны при запросе. И в этом случае мы можем включить в выборку только нужные поля. Например, выведем только порцию информации, например, значения полей address у всех документов:



Использование единицы в качестве параметра {address: 1} указывает, что запрос должен вернуть только содержание атрибута address.

В этом примере мы не конкретизируем выборку, а хотим вывести все документы, для этого оставляем первые фигурные скобки пустыми.

И обратная ситуация: мы хотим найти все поля документа, кроме свойства address. В этом случае в качестве параметра указываем 0:

db.buildings.find({}, {address: 0})

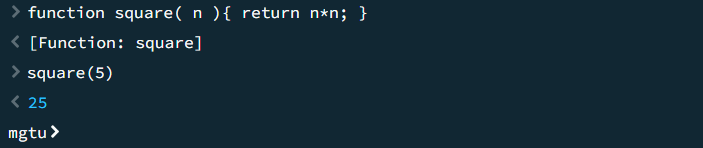
При этом надо учитывать, что даже если мы отметим, что мы хотим получить только поле address, поле \_id также будет включено в результирующую выборку. Поэтому, если мы не хотим видеть данное поле в выборке, то надо явным образом указать: {\_id: 0} (см. пример выше).

Альтернативно вместо 1 и 0 можно использовать true и false:

db.buildings.find({}, {address: true, \_id: false})

4.6. Объекты и переменные JavaScript

Кроме выполнения запросов к базе данных мы можем выполнять выражения JavaScript. Например, мы можем создать какую-нибудь функцию и применять ее:

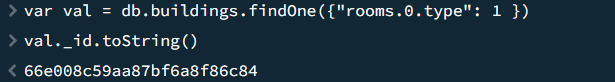


И подобные функции и выражения JavaScript мы можем применять в запросах к БД.

Используем JavaScript для получения \_id найденного документа. Используя функцию findOne, найдем документ:

var val = db.buildings.findOne({"rooms.0.type": 1 })

Документ сохраняется в переменной val. Далее, получаем \_id и ObjectId(…) преобразуем в строку:



4.7. Ограничения, пропуск и сортировка

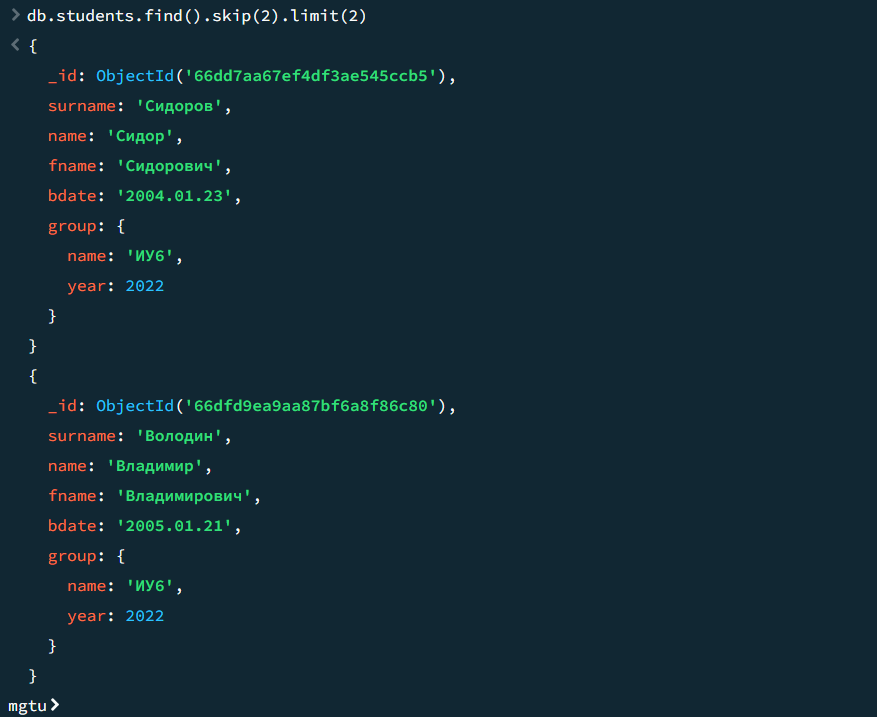
MongoDB представляет ряд функций, которые помогают управлять выборкой из БД. Одна из них - функция limit. Она задает максимально допустимое количество получаемых документов. Количество передается в виде числового параметра. Например, ограничим выборку двумя документами:



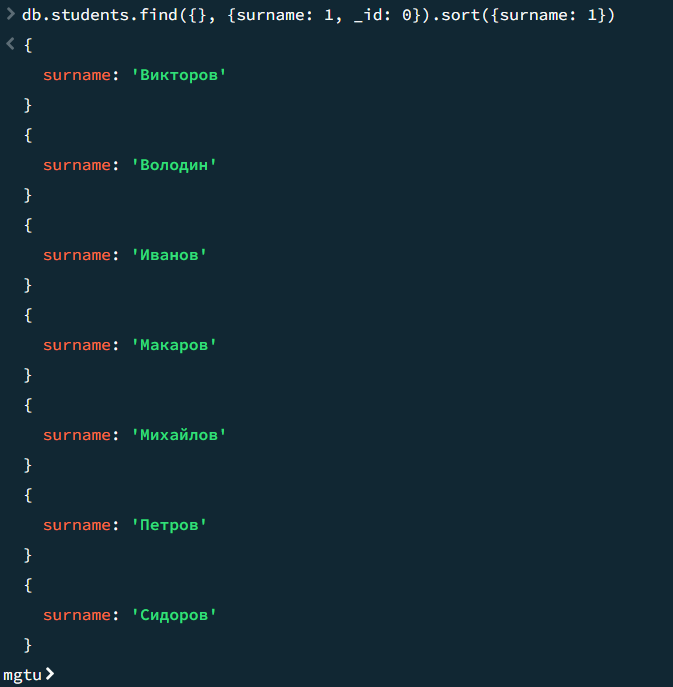
В данном случае мы получим первые два. Но что, если мы хотим произвести выборку не сначала, а пропустив какое-то количество документов? В этом нам поможет функция skip. Например, пропустим первые две записи:



Комбинируя обе функции, мы можем получить определенное количество документов, начиная с определенного документа. Например, выберем документы со 2 по 4:



MongoDB предоставляет возможности отсортировать полученный из БД набор данных с помощью функции sort. Передавая в эту функцию значения 1 или -1, мы можем указать в каком порядке сортировать: по возрастанию (1) или по убыванию (-1). Например, сортировка по возрастанию по полю surname:



5. Проектирование схемы базы данных

Модель «сущность–связь» или ER–модель[[1]](#footnote-1) представляет собой схему, составными элементами которой являются:

* **Сущность** — это реальный, либо воображаемый объект, информацию о котором необходимо хранить в базе данных. На диаграмме ER–модели сущность изображается в виде прямоугольника, содержащего имя сущности.
* **Связь** — отображаемая графически на диаграмме ассоциация между двумя (чаще всего) сущностями, или между одной и той же сущностью (рекурсивная связь).

У сущностей есть атрибуты — характеристики, которые их описывают. Например, атрибутами сущности students будут фамилия, имя, отчество и т.д. Атрибуты есть у каждого документа сущности, но у них разные значения.

В ER–модели есть три типа связей:

* **«Один-к-одному»** — один экземпляр сущности **А** связан только с одним экземпляром другой сущности **Б**. Например, коллекция «студенты» и коллекция «адреса регистрации» (если сделали две разных коллекции в БД).  
  Смоделировать такое отношение можно двумя способами:
  + прямое встраивание дочернего объекта в виде поддокумента;
  + размещение дочерних документов в отдельной коллекции и связывание через идентификатор.

Выбор способа зависит от того, как приложение получает доступ к данным, и как часто это происходит. Важен и жизненный цикл набора данных. Например, что должно произойти с документом **Б**, если будет удален документ **А**? В данном вопросе следует руководствоваться правилом.

Если документ **Б** должен быть доступен сам по себе (вне контекста родительского документа **А**), то используйте ссылку, иначе встраивайте напрямую.

**«Один-ко-многим»** — один экземпляр сущности **А** связан со множеством экземпляров другой сущности **Б**. Вторая сторона при этом всегда имеет только одну связь. Например, коллекция «группы» и коллекция «студенты». Смоделировать подобное отношение в базе также можно с помощью прямых вложений или ссылками. Возможности встраивания при этом серьезно ограничены. Например, не рекомендуется его использовать если поддокумент будет увеличиваться:

* + Размер каждого документа не может превышать 16 Мбайт.
  + Для растущего документа нужно будет выделять новое пространство, а также обновлять индексы, что влияет на производительность.

В таких ситуациях лучше предпочесть ссылки. Коллекции **А** и **Б** при этом моделируются как автономные сущности. При этом производительность чтения снижается, так как приходится выполнять второй запрос для получения сведений о сущности **Б**. Чтобы решить эту проблему используйте рациональное индексирование (для оптимизации памяти) и проекции(для снижения пропускной способности). Для выбора способа связывания можно руководствоваться правилом.

Если на стороне **Б** меньше пары сотен документов, и они не являются автономными сущностями, то их можно безопасно встроить (например, иллюстрации к книге). Если их больше пары сотен, то лучше использовать связывание по идентификатору. Если таких документов несколько тысяч, то предпочтительнее использовать ссылку на родителя. Документ **Б** ссылается на соответствующий документ **А**.

* **«Многие-ко-многим»** — множество экземпляров одной сущности связаны со множеством экземпляров другой сущности. Например, «группы» и «аудитории». В одной аудитории может обучаться несколько групп. С другой стороны, одна группа в течении недели обучается в нескольких аудиториях. Связь «многие-ко-многим» реализуется с помощью третьей коллекции, между которой и двумя исходными коллекциями организуется связь «один-ко-многим».

6. Установка ссылок в БД

Будем рассматривать организацию ссылок между двумя коллекциями: students2 и groups. Встраивание данных о группе в коллекцию students не является хорошим решением, поскольку группа является самостоятельной сущностью, на которую могут ссылаться другие сущности. Например, при составлении расписания, необходимо указать аудитории для группы.

Итак, коллекция students2 имеет вид:

{

\_id: …,

surname: …,

name: …,

fname: …,

bdate: …,

id\_group: … // \_id группы

}

А коллекция groups – вид:

{

\_id: …,

name: …,

year: …,

id\_student: … // староста группы, ссылка на students2

}

6.1. Автоматическое связывание

Используя функциональность DBRef, мы можем установить автоматическое связывание между документами. Посмотрим на примере применение данной функциональности. Вначале добавим две группы в коллекцию groups, результат сохраним в переменных, т.к. возвращаются \_id, добавленных документов:



Вставим в коллекцию students2 документ с указанием ссылки на группу:

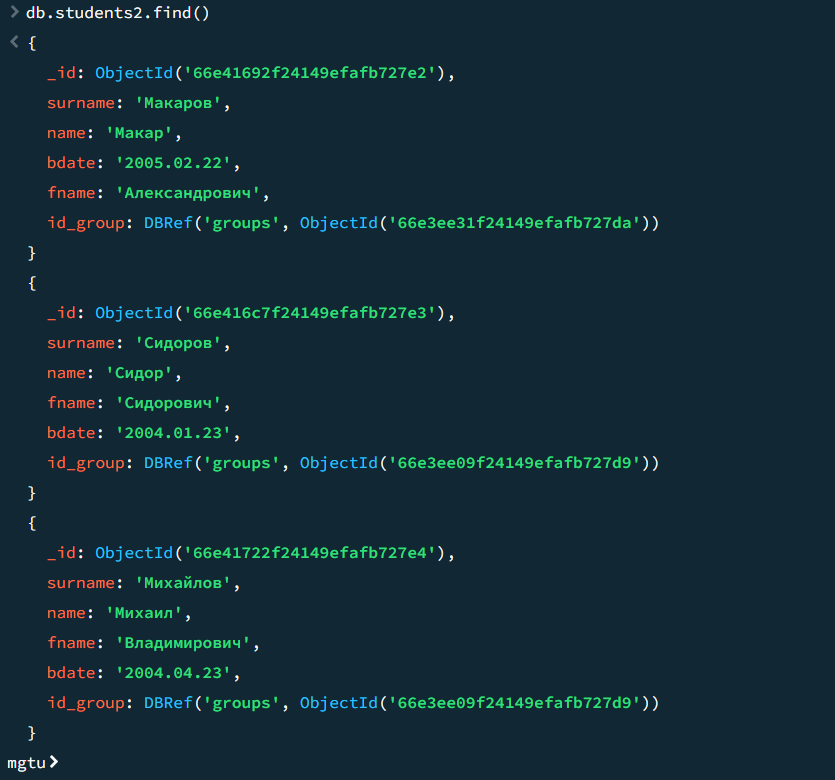


Разберем организацию ссылок между документами. Для связывания с документом grIU6\_2023 использовалось следующее выражение   
{$ref: "groups", $id: grIU6\_2023.insertedId}.

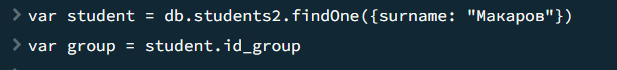
Формальный синтаксис DBRef следующий:

{$ref: <коллекции>, $id: <значение> [, $db: <бд>]}

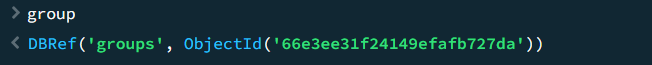
Добавим еще несколько документов в коллекцию students2. И посмотрим на результат работы:



Теперь воспользуемся результатом нашей работы. Найдем группу студента Макарова. Сначала ищем документ по фамилии.



Получаем группу студента. Объект группа имеет формат:

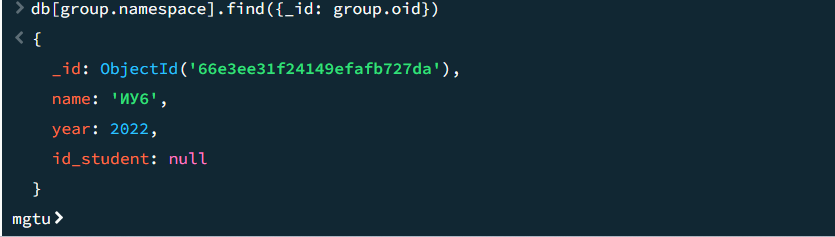


Где DBRef — это отдельный тип данных:

new DBRef(namespace, oid, db)

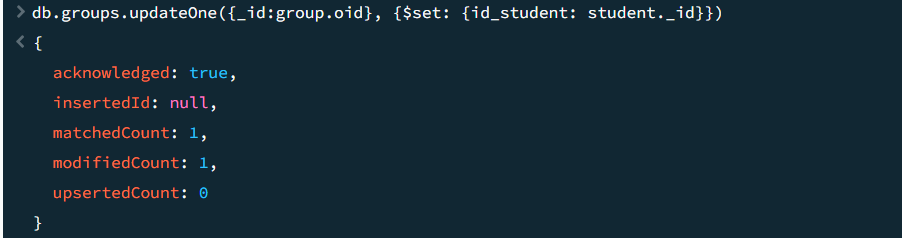
| **Имя** | **Тип** | **Описание** |
| --- | --- | --- |
| namespace | string | название коллекции |
| oid | ObjectID | ссылка ObjectID |
| db | string | необязательное имя базы данных, если не указано,   ссылка будет локальной для текущей базы данных |

Находим группу:

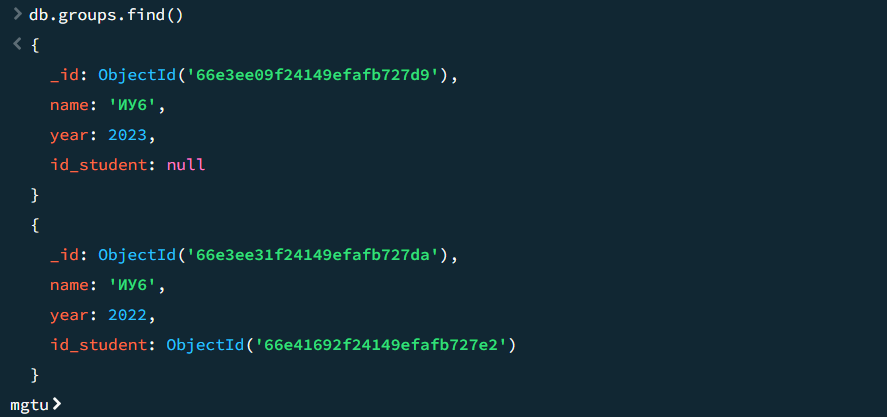


6.2. Ручная установка ссылок

Ручная установка ссылок сводится к присвоению значения поля \_id одного документа полю другого документа. Занесем в документ коллекции groups в атрибут id\_student (староста) студента Макарова:



Здесь мы воспользовались объектами student и group, полученными в предыдущем разделе. Посмотрим на то, что получили:



Найти документ (студента) по \_id не составляет никакой сложности.

7. Практическая часть

В данной лабораторной работе вам предстоит разработать фрагмент базы данных системы, предметную область которой кратко описана в Приложении. Конкретную предметную область можно получить, взяв вариант, совпадающий с вашим порядковым номером в списке группы.

Для разработки ER–модели можно использовать любую предназначенную для этой цели программу.

Надо сказать, что приведенные схемы весьма общие, а иногда и не очень корректные, более того MongoDB не подходит для этих задач, поэтому вы можете изменить их по вашему усмотрению, оставаясь, конечно, в рамках предметной области.

Создать БД, коллекции, используя три варианта связывания: встраиваемые документы, массив ссылок на дочерние документы и ссылки дочерних документов на родительские. Ввести данные в не менее 100 документов для целевых сущностей, обеспечивая связь между коллекциями.

Приветствуется использования языков программирования для работ c MongoDB.

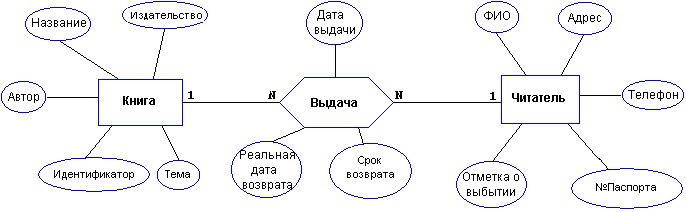
8. Список использованных источников

1. MongoDB: <https://www.mongodb.com/>
2. Шеннон Брэдшоу, Йон Брэзил, Кристина Ходоров  
   MongoDB: полное руководство. Мощная и масштабируемая система управления базами данных / пер. с англ. Д. А. Беликова – М.: ДМК Пресс, 2020. – 540 с.: ил.

9. Приложение. Схемы предметных областей

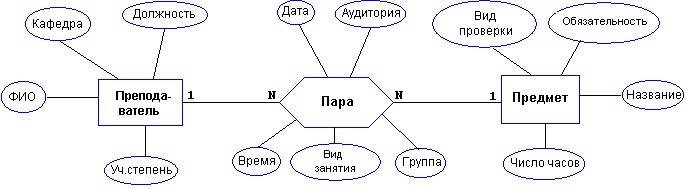
Вариант №1

Предметная область для практических заданий: **Библиотека**



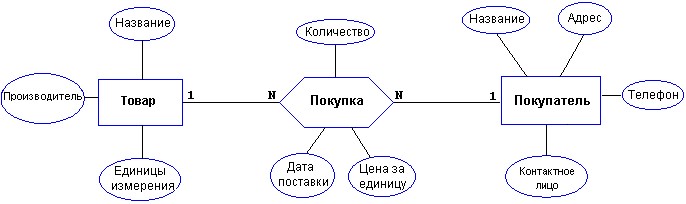
Вариант №2

Предметная область для практических заданий: **Университет**



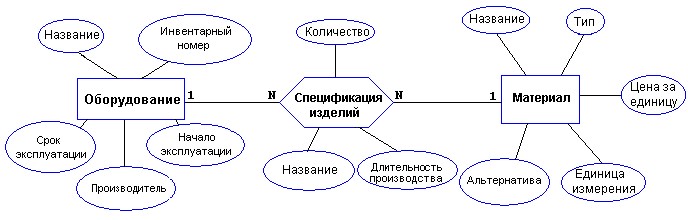
Вариант №3

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



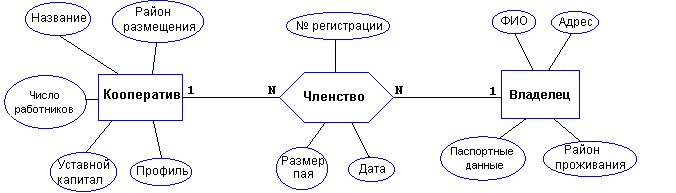
Вариант №4

Предметная область для практических заданий: **Производство**



Вариант №5

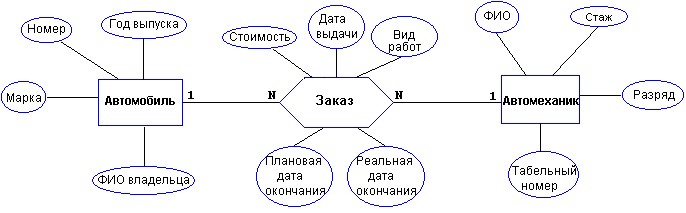
Предметная область для практических заданий: **Кооперативы**



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

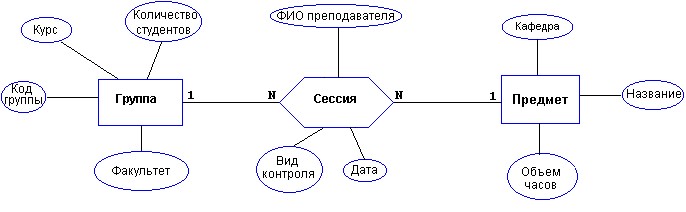
Вариант №6

Предметная область для практических заданий: **Автомастерская**



Вариант №7

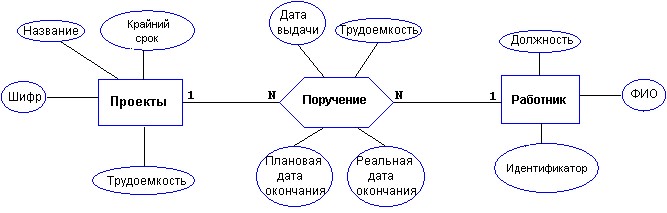
Предметная область для практических заданий: **Сессия**



Вид контроля - зачет, экзамен.

Вариант №8

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

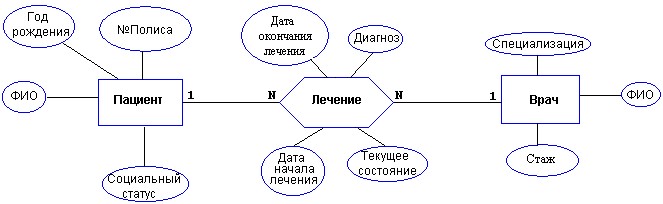


Категория дисциплины - гуманитарная, математическая, компьютерная и т.д.

Вид контроля - зачет, экзамен.

Вариант №9

Предметная область для практических заданий: **Поликлиника**

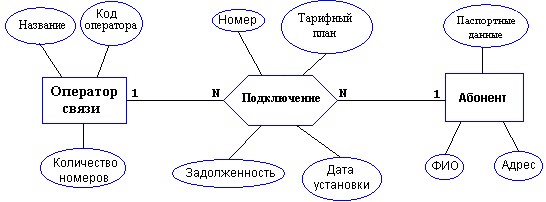


Текущее состояние - средней тяжести, тяжелое, направлен в стационар, умер. Социальный статус пациента - учащийся, работающий, врем. неработающий, инвалид, пенсионер

Специализация врача - терапевт, хирург и т.п.

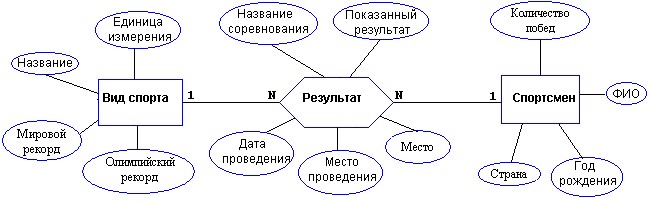
Вариант №10

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



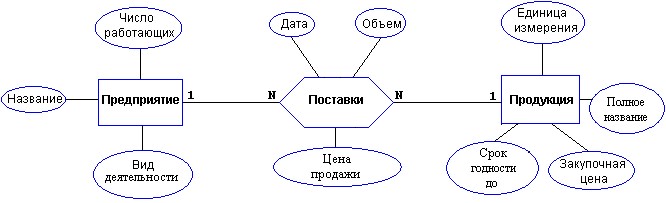
Вариант № 11

Предметная область для практических заданий: **Спорт**



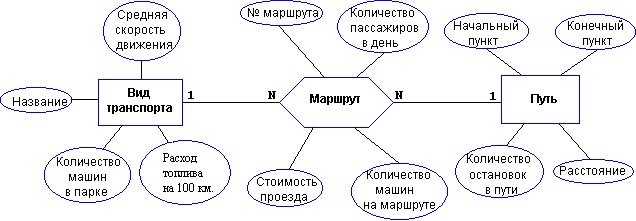
Вариант №12

Предметная область для практических заданий: **Поставки**



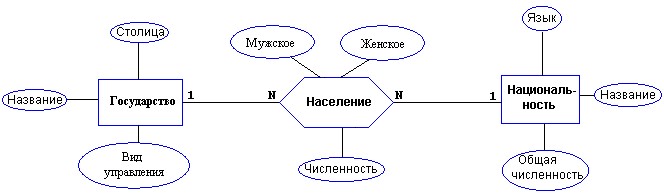
Вариант №13

Предметная область для практических заданий: **Транспорт**



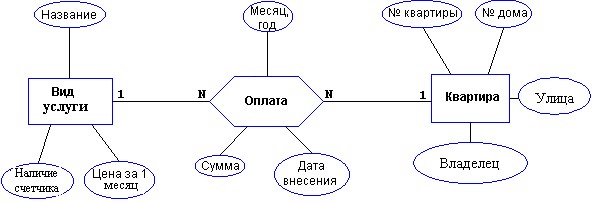
Вариант № 14

Предметная область для практических заданий: **География**



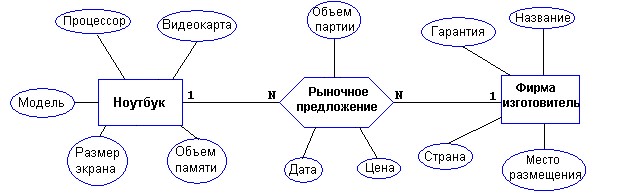
Вариант №15

Предметная область для практических заданий: **Домоуправление**



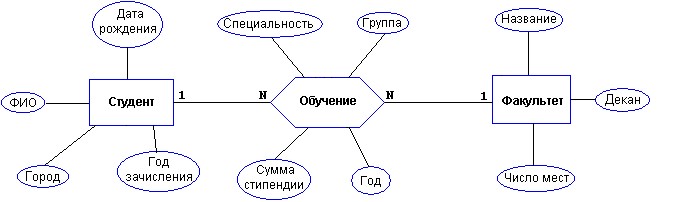
Вариант №16

Предметная область для практических заданий: **Ноутбуки**



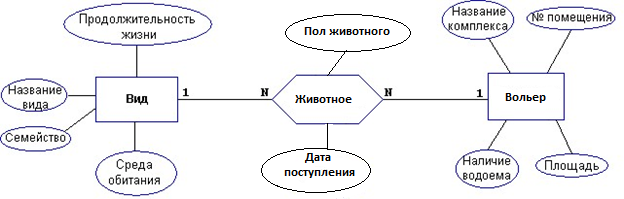
Вариант №17

Предметная область для практических заданий: **Деканат**



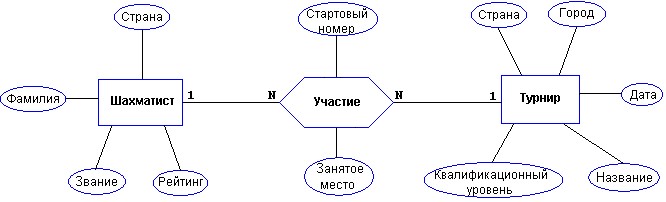
Вариант №18

Предметная область для практических заданий: **Зоопарк**



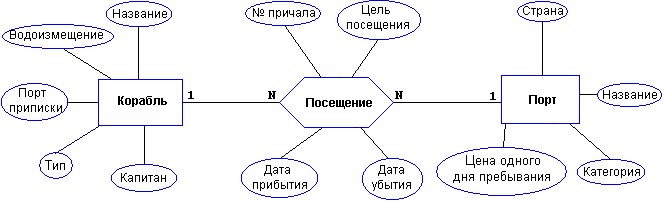
Вариант №19

Предметная область для практических заданий: **Шахматы**



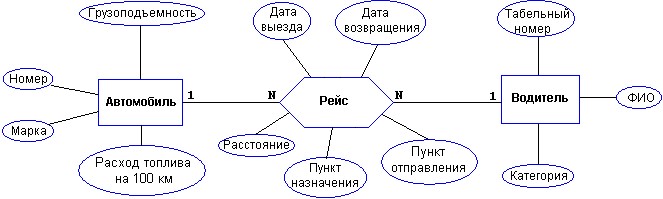
Вариант №20

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



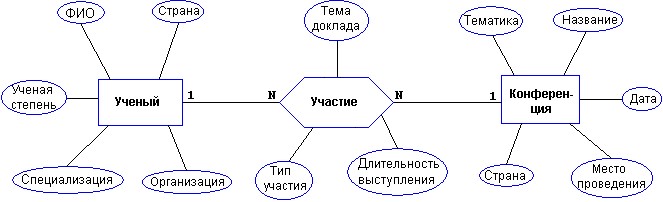
Вариант №21

Предметная область для практических заданий: **Грузоперевозки**



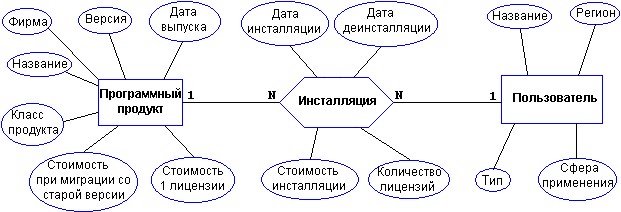
Вариант №22

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



Вариант №23

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

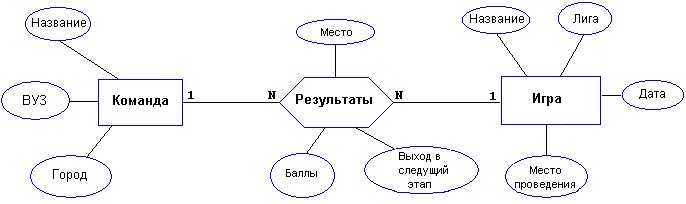


Класс: ОС, сервер приложений, СУБД, Web-сервер, система программирования и т.д.

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

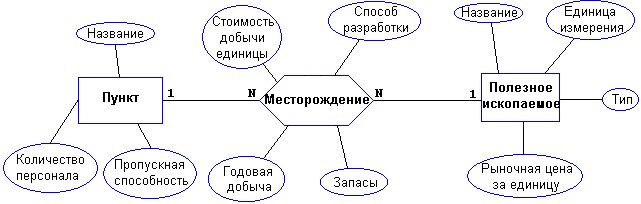
Вариант №24

Предметная область для практических заданий: **КВН**



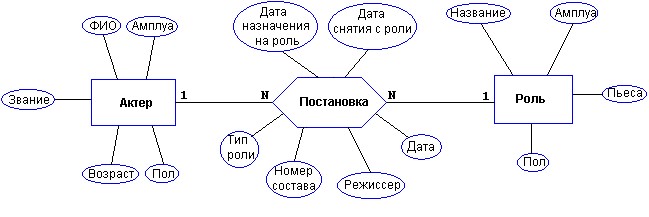
Вариант №25

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



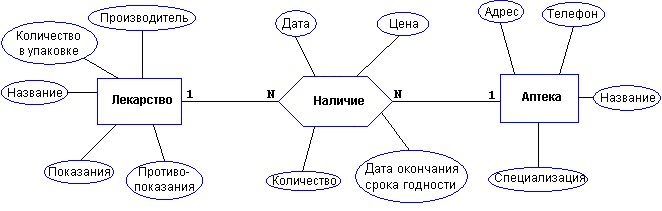
Вариант №26

Предметная область для практических заданий: **Театр**



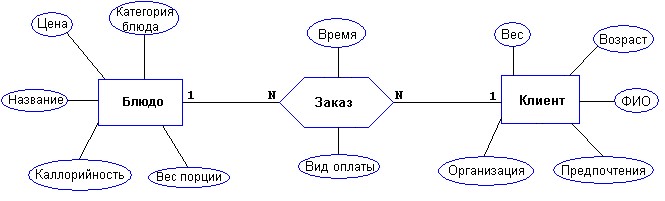
Вариант №27

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



Вариант №28

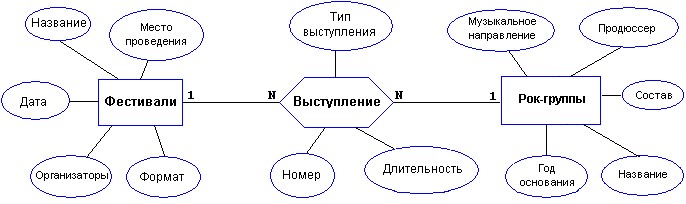
Предметная область для практических заданий: **Столовая**



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

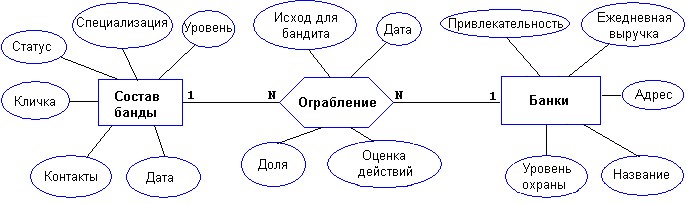
Вариант №29

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



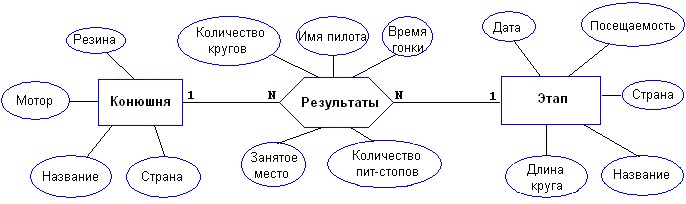
Вариант №30

Предметная область для практических заданий: **ОПГ**



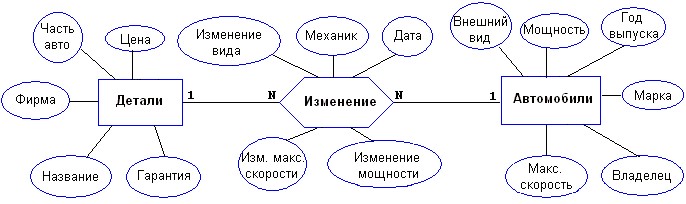
Вариант №31

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



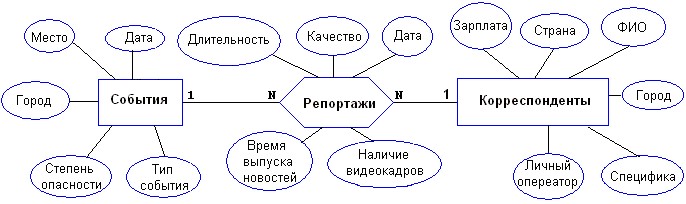
Вариант №32

Предметная область для практических заданий: **Тюнинг**



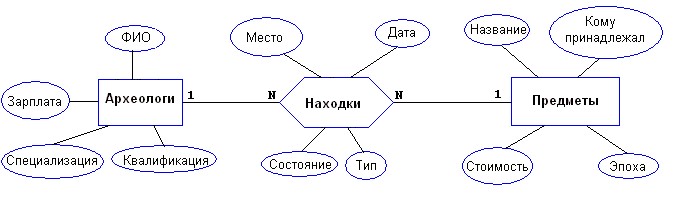
Вариант №33

Предметная область для практических заданий: **Тележурналистика**



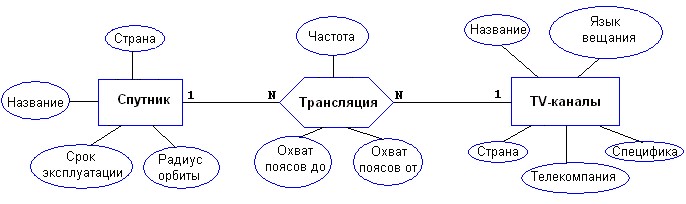
Вариант №34

Предметная область для практических заданий: **Археология**



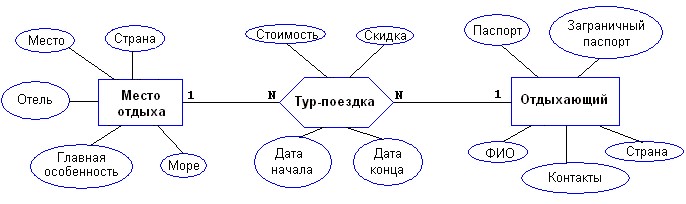
Вариант №35

Предметная область для практических заданий: **Телевещание**



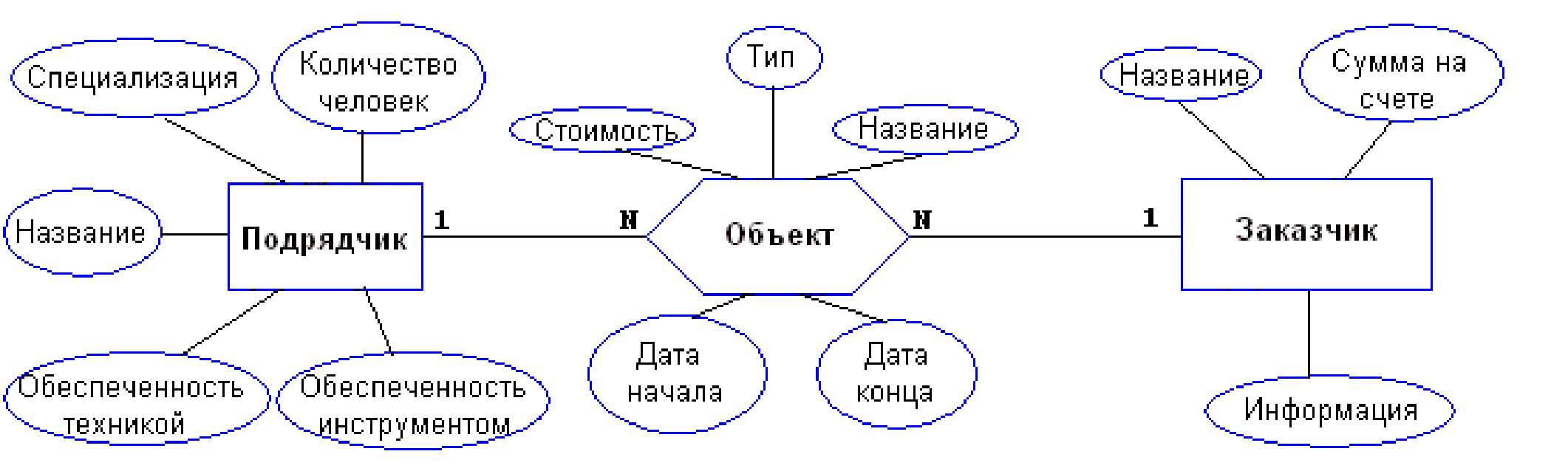
Вариант №36

Предметная область для практических заданий: **Турфирма**

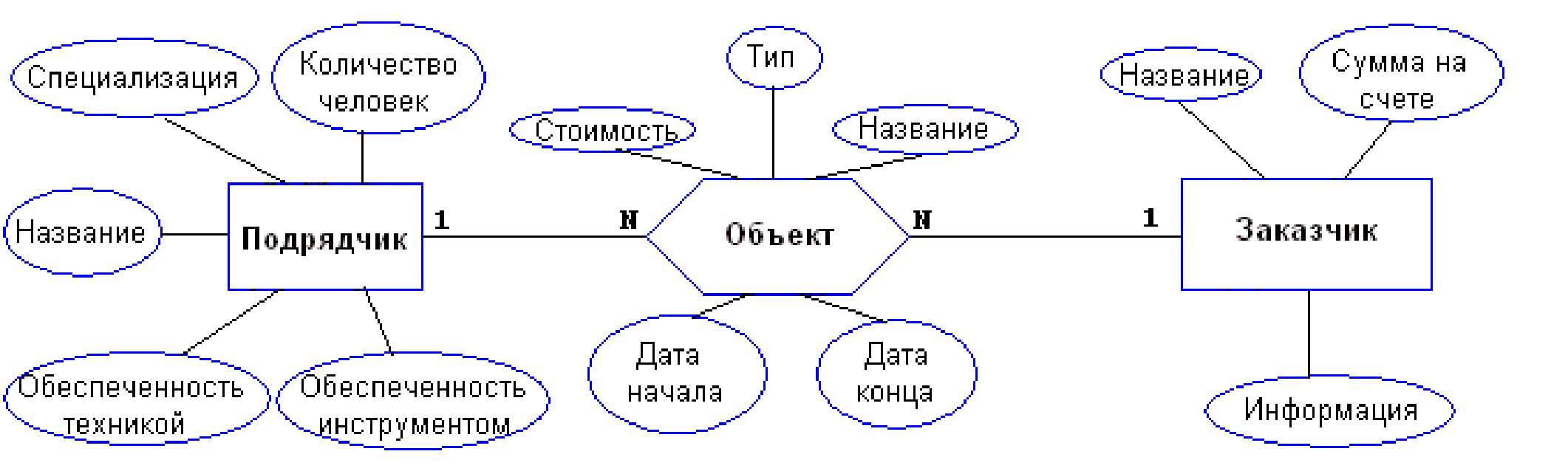


Вариант №37

Предметная область для практических заданий: **Строительство**



Тип объекта: промышленный, частный, специальный, хозяйственный и.т.д

Вариант №38

Предметная область для практических заданий: **Искусство**

Тип произведения: скульптура, живопись, литье, графика и т.д.

Тип места хранения: частная коллекция, музей, галерея, неизвестно (тогда все остальные параметры пустые) и т.д.

1. Надо отметить, модель «сущность–связь» разработана для реляционный СУБД. Однако, она будет полезна и для проектирования NoSQL СУБД. [↑](#footnote-ref-1)