МДК.13.02 Разработка серверной и интерфейсной частей приложения (Backend, Frontend разработка на языках Python, C++, Java)

Практическое занятие № 1

Тема практического занятия: Основы MongoDB. Создание, удаление, обновление, чтение данных.

Цель практического занятия: Освоить основные способы работы с mongo db. Научиться работать с документами и коллекциями.

В результате выполнения данной работы обучающийся должен
уметь:

- устанавливать MongoDB на своем компьютере и настраивать сервер MongoDB;
- вставлять новые документы в коллекцию MongoDB, предоставляя необходимую информацию;
- обновлять информацию в существующих документах коллекции, например, изменять значения полей;
- удалять данные из коллекции, в том числе по определенным критериям

знать:

- особенности нереляционных бд;
- основные команды для работы с данными, особенности использования MongoDB;
- основные объекты экосистемы MongoDB (база данных, документ, коллекция и т.п.)

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

База данных – это поименованная совокупность структурированных, хранящихся вместе данных, при наличии такой минимальной избыточности, при которой возможно ее использование одним или несколькими приложениями.

Существуют реляционные и нереляционные базы данных.

Реляционные базы данных представляют собой совокупность таблиц и связей между ними. Данные содержатся непосредственно в строках таблицы. Подробнее о реляционных базах данных вы узнаете чуть позже.

Нереляционные базы данных используют различные модели данных для работы с данными и управления ими. Таки БД подходят для приложений, использующих большой объем данных или требующих более гибкой модели работы с данными.

Основные плюсы использования нереляционных БД:

1. Гибкая структура, которая позволяет работать с различного рода данными (видео, изображения, документы), с данными не имеющей четко выраженной структуры.
2. Высокая производительность в определенном наборе задач
3. Высокий уровень горизонтальной масштабируемости (можно с легкостью разбить бд на несколько узлов и распределить нагрузку)

Таким образом, наилучшим кейсом применения нереляционных баз данных является работа с данными, структура которых не определена или может измениться.

Для работы мы будем использовать MongoDB – документо-ориентированную систему управления базами данных.

Вместо строк как в реляционных бд MongoDB хранит документы. Документы представляют собой сложные по структуре наборы ключей и значений. Для идентификации документов используется уникальный идентификатор _id. По умолчанию mongo db сам генерирует этот идентификатор если он не указан явно.

Вместо привычных таблиц MongoDB хранит коллекции. Коллекция – это набор документов. При этом структура документов может быть совершенно различна.

Данные в БД хранятся в формате BSON (binary JSON), что позволяет выполнять быстрое чтение данных из БД.

Основные команды для работы с Mongo DB в практическом занятии 1:

- use [database_name] – установка бд в качестве текущей, если такой бд нет, то она автоматически создается
- show dbs – вывести список всех имеющихся баз данных
- show collections – список всех коллекций в текущей бд
- db.stats(), db.<collection_name>.stats() – посмотреть статистику по текущей бд или коллекции в текущей бд
- db.dropDatabase() – удаление бд
- db.createCollection([collection_name]) – создание коллекции
- db.[collection_name].drop() – удаление коллекции

- db.<collection_name>.insertOne(<document>) – добавление одного документа в коллекцию
- db.<collection_name>.insertMany([doc1, doc2…]) – добавление нескольких документов в коллекцию
- load([file_path]) – загрузка документов в текущую базу из файла
- db.<collection_name>.deleteOne(filter) – удаляет первый документ из коллекции, который соответствует фильтру
- db.<collection_name>.deleteMany(filter) – удаляет все документы из коллекции, которые соответствуют фильтру
- db.<collection_name>.replaceOne(filter, update, options) – обновление документа по фильру. Заменяет документ на значение update, используя дополнительные параметры options.
- db.<collection_name>.updateOne(filter,update), db.<collection_name>.updateMany(filter, update) – обновление одного или нескольких документов. update принимает то, какие данные следует обновить или заменить.
- db.<collection_name>.find() – выборка элементов из коллекции

Задание:

Подключиться к серверу и выполнить основные базовые операции MongoDB создания, чтения, обновления и удаления данных.

Технология выполнения работы:

1. Подключитесь к локальному серверу mongo db и создайте базу данных с именем warehouse (склад)
2. Выведите в консоль все базы данных и проверьте, что она создалась
3. В базе данных создайте коллекции products (информация о товаре на складе), employees (информация о сотрудниках склада)
4. Выведите все коллекции в бд
5. Заполните коллекцию products документами используя команды insertOne() и insertMany() информацией о товаре – наименование, описание, артикул, цена, количество или вес (в зависисмости от товара), единица измерения товара, категории товара (может быть несколько). Добавьте не менее 8 документов.
6. Заполните коллекцию employees документами из файла с помощью команды load(). Содержание документов выберите сами, но она должна отражать информацию о сотруднике. Добавьте не менее 8 документов.
7. Выведите содержимое обеих коллекций
8. Выведите статистику по базе данных и коллекциям в ней.
9. Удалите из коллекции products все документы, в которых цена товара больше 100.
10. Замените с помощью команды replaceOne() одну запись из коллекции employees. Фильтр выберите произвольный.
11. Обновите все документы в коллекции products в которых отсутствует описание, добавьте ключу описание значение «Нет описания товара»

12. Обновите один документ в коллекции employees. Фильтр и значение для обновления выберите сами.
13. Удалите все созданные коллекции
14. Удалите базу данных warehouse

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Освоены принципы работы с mongo db
- Создана база данных и коллекции
- Выполнены все запросы к БД
- Сохранены скриншоты с запросами и результатами их выполнения в консоли
- Даны ответы на контрольные вопросы

Контрольные вопросы:

1. Что такое нереляционные БД?
2. Какие плюсы имеют нереляционные БД?
3. Что такое MongoDB?
4. Дайте определение понятиям документ, коллекция.
5. В каком формате данные хранятся в MongoDB?
6. Расскажите об основных командах, используемых при работе с MongoDB

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 2

Тема практического занятия: Выполнение запросов на выборку. Фильтрация, лимитизация, аггрегация и ранжирование данных. Оптимизация запросов.

Цель практического занятия: Освоить методы написания запросов к БД. Научиться оптимизировать запросы. Получить навыки работе в графической оболочке MongoDB Compass

В результате выполнения данной работы обучающийся должен уметь:

- выполнять запросы поиска документов,
- фильтровать, ранжировать, лимитизировать, аггрегировать результаты выборки.
- работать с MongoDB Compass

знать:

- команды, необходимые для написания запросов к БД.
- методы оптимизации запросов.

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Основной задачей при взаимодействии приложений с бд является получение данных. Для этого MongoDB предоставляет методы для поиска документов и работы с результатами поиска.

Определения:

Выборка – набор документов из бд.

Фильтрация – поиск документов, удовлетворяющих фильтру (условию).

Лимитизация – ограничение количества документов в выборке.

Ранжирование (сортировка) – распределение данных по возрастанию или убыванию в зависимости от указанных параметров сортировки.

Аггрегация – подсчет скалярных величин на основе результатов выборки (например среднее арифметическое или сумма).

Программист может написать запрос к БД и если нужно применить методы работы с выборкой – фильтрацию, лимитизацию, ранжирование или аггрегацию.

Так как часто количество данных бывает большим, то запросы должны выполняться быстро. Для этого могут быть использованы следующие методы оптимизации запросов.

- Для частых запросов можно создать индексы
- При необходимости лимитизации обязательно указывайте ее в запросе
- Получайте только необходимые данные

Команды для выполнения практического занятия:

- db.<collection_name>.find(filter) – поиск и фильтрация данных
- .sort(parameters) – сортировка данных
- .skip(n) – пропуск n значений
- .limit(n) – лимитизация выборки
- .distinct() – уникальные документы
- .count() – количество документов в выборке
- .min() – минимальное значение для попадания в выборку
- .max() – максимальное значение для попадания в выборку
- .aggregate() - аггрегация

Для написания запросов используем графическую оболочку MongoDB Compass в которой можно удобно писать запросы и управлять бд.

Задание:

Провести скачивание данных и создание базы данных. Выполнить запросы на выборку, аггрегирование, лимитизацию данных. Выполнить экспорт результатов выборки.

Технология выполнения работы:

1. Скачайте json файл с данными о ресторанах по ссылке mongodb-json-files/datasets/restaurant.json at master · ozlerhakan/mongodb-json-files (github.com)
2. Установите и откройте MongoDB Compass. Создайте базу данных с произвольным именем, а в ней коллекцию restaurants
3. В коллекцию restaurants с помощью импортируйте данные о ресторанах из скачанного файла.
4. В интерфейсе MongoDB Compass откройте консоль и выполните запрос на получение всех документов. Проверьте, что все работает корректно.
5. Выполните следующие запросы в консоли MongoDB Compass
6. Посчитать сколько всего документов в коллекции
7. Вывести первые 10 документов в коллекции
8. Выведите 10 документов начиная с 7-го
9. Найдите все рестораны с рейтингом 4 и выше. Посчитайте их количество.
10. Найдите все рестораны с рейтингом от 2 до 5. Посчитайте их количество.

11. Найдите все рестораны с китайской кухней (type_of_food – Chinese) и отсортируйте их в порядке убывания рейтинга.
12. Найдите все рестораны, в которых в названии есть буква «е”. Отсортируйте их в алфавитном порядке, а потом в порядке возрастания рейтинга.
13. Найдите все рестораны, в которых есть ссылка на меню. Отсортируйте их в порядке убывания postcode и выведите первые 5.
14. Найдите все рестораны китайской кухни с рейтингом выше 3, посчитайте среднее арифметическое их рейтинга.
15. Выполните следующий запрос с помощью интерфейса: найти все рестораны, в адресе которых есть слово Road
16. Посмотрите информацию о плане запроса
17. Произведите экспорт полученной коллекции через графический интерфейс. Поле _id не включайте в список полей экспорта. В качестве формата экспортируемого файла выберите csv.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Выполнены запросы
- Изучены команды и методы оптимизации запросов
- Произведен экспорт данных запроса в csv файл
- Создан отчет со скриншотами проделанной работы
- Даны ответы на контрольные вопросы

Контрольные вопросы:

1. Дайте определение следующих понятий – выборка, фильтрация, аггрегация, лимитизация, ранжирование (сортировка).
2. Перечислите основные команды, используемые при поиске документов
3. Перечислите методы оптимизации запросов
4. Что такое MongoDB Compass?
5. Как производить импорт и экспорт файлов, делать запросы в Mongo DB Compass?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.

4. 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.

Практическое занятие № 3

Тема практического занятия: Управление индексами. Ускорение запросов при помощи индексов.

Цель практического занятия: Освоить методы ускорения запросов и создания индексов в Mongo DB

В результате выполнения данной работы обучающийся должен уметь:

- создавать индексы в mongo db;
- управлять индексами;
- ускорять запросы к базе данных

знать:

- команды, необходимые для создания индексов;
- принципы работы индексов;
- виды индексов

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Красота базы данных заключается в индексации, т.е. извлечении данных за короткий промежуток времени без повторения всего набора данных. MongoDB - это база данных типа документов NoSQL, которая следует за индексацией. Индексы упрощают поиск в коллекции при ограниченном количестве документов. Двоичное дерево - это структура данных, используемая индексом. В документах поле _id является индексом по умолчанию, который автоматически создается MongoDB, и нам не разрешается удалять этот индекс. Если мы попытаемся удалить этот индекс с помощью метода dropIndexe(), это выдаст нам ошибку. И значение этого поля однозначно отличает документ от других.

Создайте индекс:

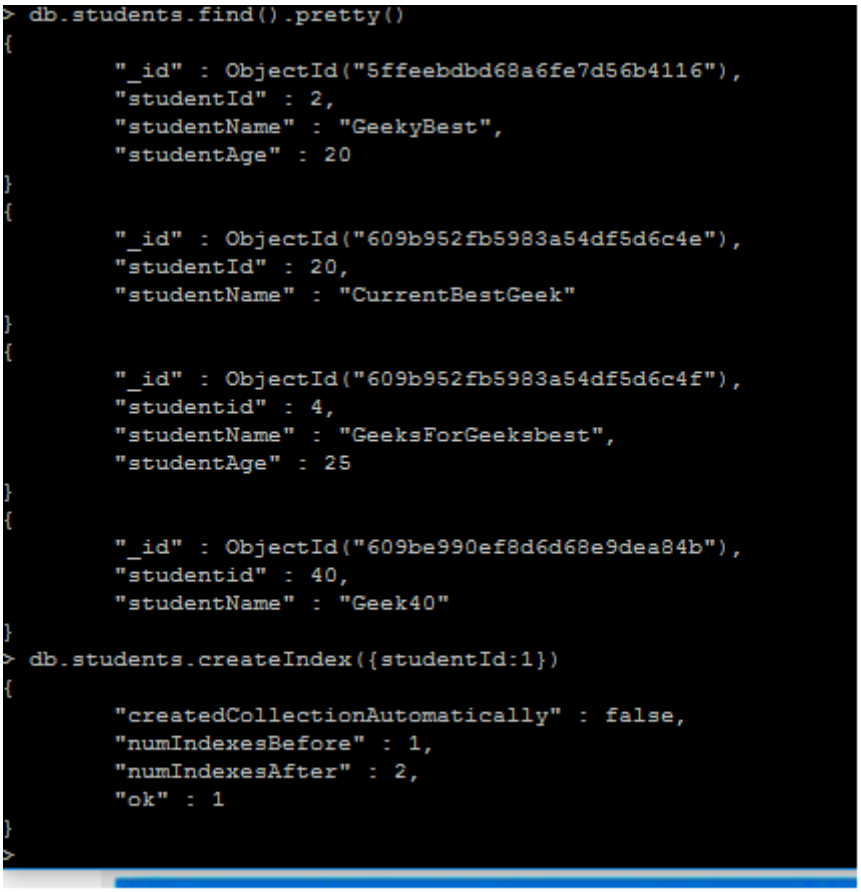
Наряду с индексом по умолчанию, мы можем создавать индексы самостоятельно, используя метод с, который используется по умолчанию. Этот метод создает один или несколько индексов для указанных коллекций.

Синтаксис :
db.Collection.name.createIndex(
ключи: {Имя_поля:1 /-1},
параметры: <документ>,
Запрос на фиксацию: <строка или целое число>
)
Пример:
В следующих примерах мы работаем с:
База данных: *gfg*
Коллекция: *студенты*
Документ: *четыре документа, содержащие сведения об учениках*
db.students.createIndex({studentsId:1})

Сначала на изображении ниже мы находим сведения о коллекции. Итак, здесь коллекция называется students, в которой мы хотим создать индекс по столбцу «StudentID”.

Примечание: в MongoDB чувствителен к регистру. Таким образом, StudentID и studentid обрабатываются по-разному.

При выполнении нашего запроса мы получили сообщение «ok», означающее, что индекс создан, а также «numIndexesAfter»: 2»один индекс уже доступен (индекс по умолчанию) и, следовательно, количество увеличено на 2.



1.1 Типы индексов

MongoDB предоставляет различные типы индексов, которые используются в зависимости от типа данных или запросов. Индексы, поддерживаемые MongoDB, следующие:

1. Индекс одного поля: индекс одного поля означает индекс для одного поля документа. Этот индекс полезен для выборки данных как в порядке возрастания, так и в порядке убывания.

Синтаксис:

db.students.createIndex({«<Имя_поля>” : <1 или -1>});

Здесь 1 обозначает поле, указанное в порядке возрастания, а -1 - в порядке убывания.

Пример:

db.students.createIndex({studentsId:1})

В этом примере мы создаем единый индекс для поля studentsId, и поле задается в порядке возрастания.

```
> db.students.find().pretty()
{
        "_id" : ObjectId("5ffeebdbd68a6fe7d56b4116"),
        "studentId" : 2,
        "studentName" : "GeekyBest",
        "studentAge" : 20
}
{
        "_id" : ObjectId("609b952fb5983a54df5d6c4e"),
        "studentId" : 20,
        "studentName" : "CurrentBestGeek"
}
{
        "_id" : ObjectId("609b952fb5983a54df5d6c4f"),
        "studentid" : 4,
        "studentName" : "GeeksForGeeksbest",
        "studentAge" : 25
}
{
        "_id" : ObjectId("609be990ef8d6d68e9dea84b"),
        "studentid" : 40,
        "studentName" : "Geek40"
}
> db.students.createIndex({studentId:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

2. Составной индекс: Мы можем объединить несколько полей для составного индексирования, и это поможет при поиске или фильтрации документов таким образом. Или, другими словами, составной индекс - это индекс, в котором одна структура индекса содержит несколько ссылок.

Синтаксис:

db.<collection>.createIndex({ <field1>: <type>, <field2>: <type2>, ... })

Здесь мы можем объединить требуемые поля в этот шаблон. Также значение этих полей равно 1 (для порядка возрастания) или -1 (для порядка убывания).

Примечание: Составные индексы могут иметь одно хэшированное индексное поле, но хэшированные индексы требуют функции хэширования для вычисления хэша значения индексного поля.

Пример:

Здесь мы создаем составной индекс для studentAge: 1, studentName:1

db.students.createIndex({studentAge: 1, studentName:1})

```
> db.students.createIndex({studentAge: 1, studentName:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

db.students.find().sort({«studentAge»:1,»studentName»:1}).pretty()

Здесь мы используем функциональность сортировки на основе полей «studentAge", за которыми следуют «studentName», и, следовательно, на изображении ниже, хотя есть 2 документа, соответствующих «studentAge = 25», поскольку studentName является дополнительным значением, в качестве второго документа отображается studentName со значением «Geek40», и только после этого, в качестве третьего документа, отображается studentName со значением «GeeksForGeeksbest". Следовательно, иногда возникает необходимость в создании составных индексов, когда мы хотим иметь более высокий уровень фильтрации.

```
> db.students.find().sort({"studentAge":1,"studentName":1}).pretty()
{
        "_id" : ObjectId("5ffeebdbd68a6fe7d56b4116"),
        "studentId" : 2,
        "studentName" : "GeekyBest",
        "studentAge" : 20
}
{
        "_id" : ObjectId("609be990ef8d6d68e9dea84b"),
        "studentName" : "Geek40",
        "studentId" : 40,
        "studentAge" : 25
}
{
        "_id" : ObjectId("609b952fb5983a54df5d6c4f"),
        "studentName" : "GeeksForGeeksbest",
        "studentAge" : 25,
        "studentId" : 4
}
{
        "_id" : ObjectId("609b952fb5983a54df5d6c4e"),
        "studentId" : 20,
        "studentName" : "CurrentBestGeek",
        "studentAge" : 30
}
>
```

3. Индекс с несколькими ключами: MongoDB использует индексы с несколькими ключами для индексации значений, хранящихся в массивах. Когда мы индексируем поле, содержащее значение массива,

MongoDB автоматически создает отдельный индекс для каждого значения, присутствующего в этом массиве. Используя эти многоключевые индексы, мы можем легко найти документ, содержащий массив, путем сопоставления элементов. В MongoDB вам не нужно явно указывать индекс с несколькими ключами, потому что MongoDB автоматически определяет, создавать ли индекс с несколькими ключами, если индексируемое поле содержит значение массива.

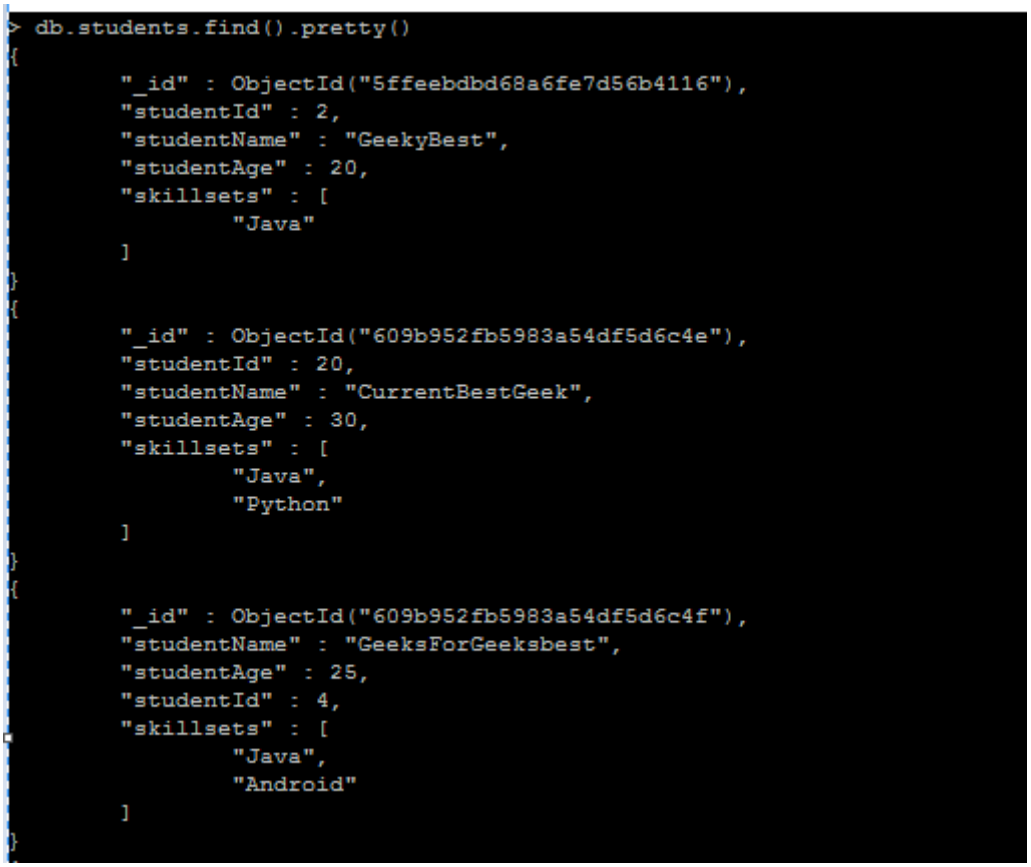
Синтаксис:

db.<коллекция>.createIndex({ <поле>: <тип>})

Здесь значение поля равно 1 (для порядка возрастания) или -1 (для порядка убывания).

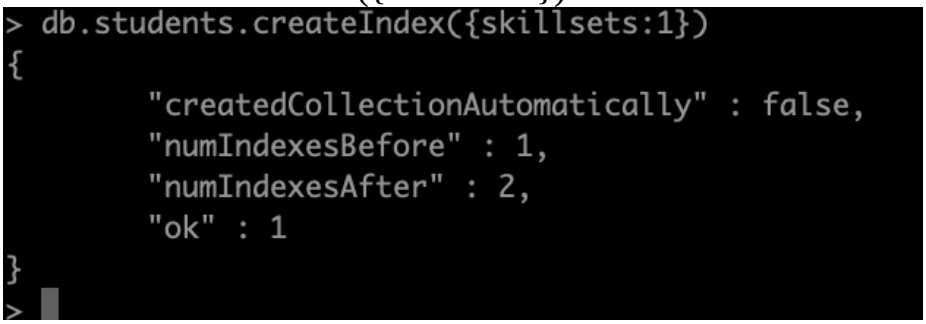
Пример:

В коллекции students у нас есть три документа, содержащих поля массива.



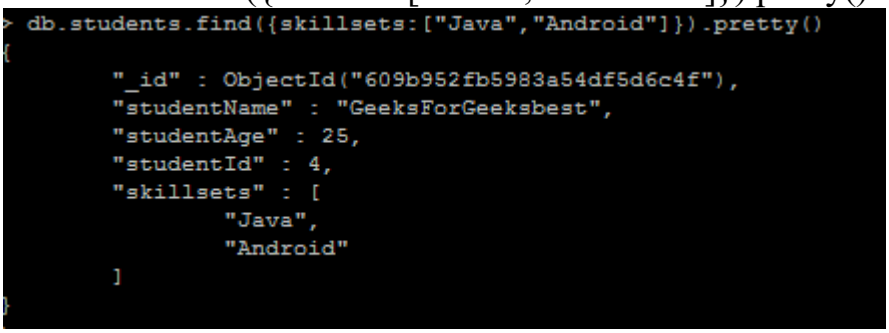
Теперь мы создаем многоключевой индекс:

db.students.createIndex({skillsets:1})



Теперь мы просматриваем документ, содержащий наборы навыков: [«Java”, «Android”]

db.students.find({skillsets:[«Java», «Android»]}).pretty()



4. Геопространственные индексы: Это важная функция MongoDB. MongoDB предоставляет два геопространственных индекса, известных как 2d-индексы и 2d-индексы сфер, используя эти индексы, мы можем запрашивать геопространственные данные. Здесь 2d-индексы поддерживают запросы, которые используются для поиска данных, хранящихся в двумерной плоскости. Он поддерживает только данные, которые хранятся в устаревших парах координат. В то время как индексы 2d sphere поддерживают запросы, которые используются для поиска данных, хранящихся в сферической геометрии. Он поддерживает данные, которые хранятся в устаревших парах координат, а также объекты GeoJSON. Он также поддерживает запросы, такие как запросы на включение, пересечение, близость и т.д.

Синтаксис индексов 2d сфер:

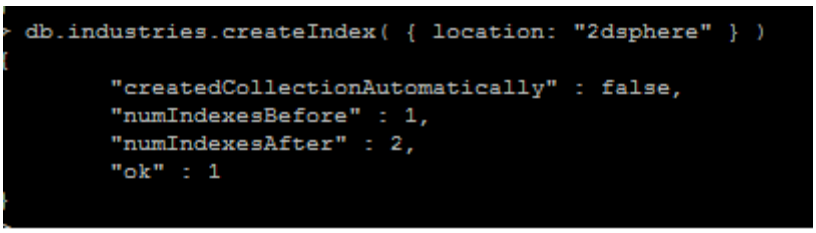
db.<collection>.createIndex({ <Locationfield>: «2dsphere”})

Пример:

Предположим, что доступные данные относятся к «отраслям”

Теперь давайте создадим индекс 2d сферы в поле location:

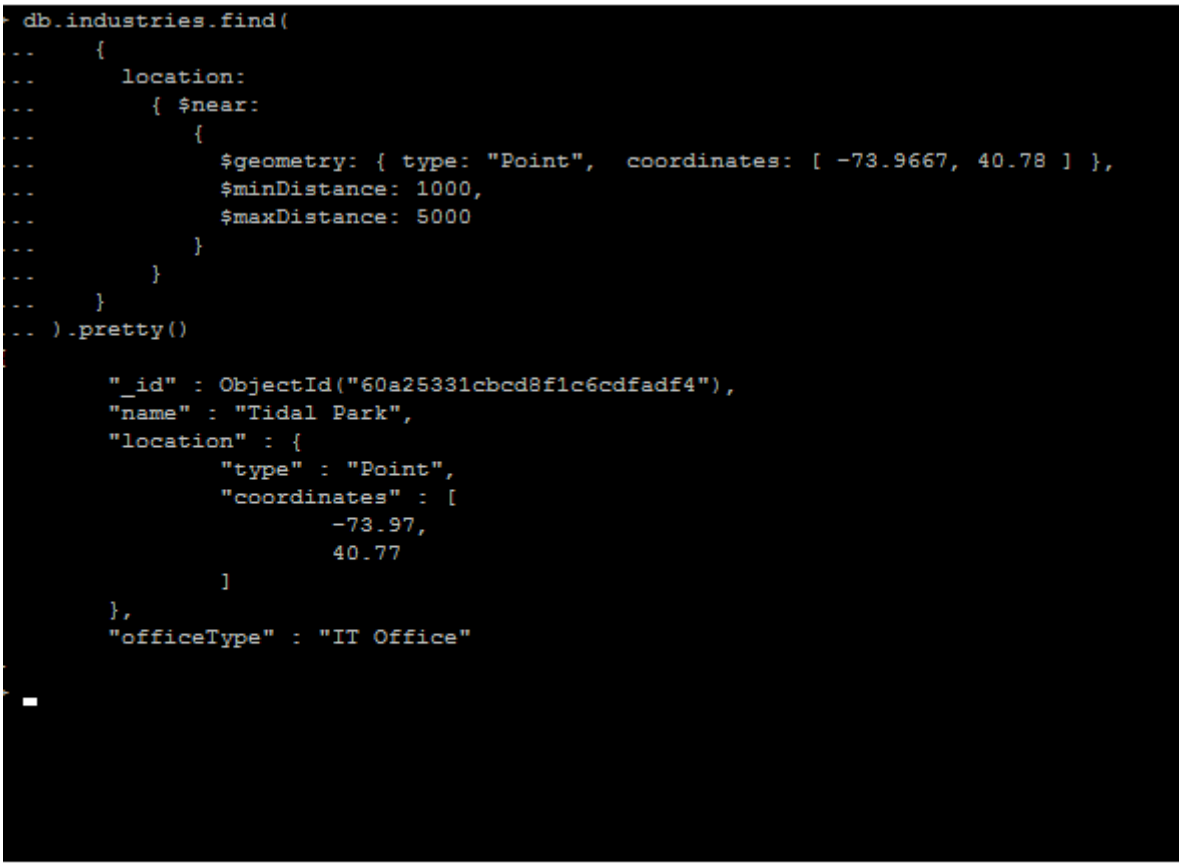
db.industries.createIndex({location:»2dsphere»})



Теперь, при выполнении приведенного ниже запроса, мы получаем

```
db.industries.find(
{
   location:
      {$near:
         {
            $geometry:{type: «Point», coordinates:[-73.9667, 40.78]},
            $minDistance:1000,
            $maxDistance: 5000
         }
      }
   }
}.pretty()
```

Здесь оператор **»$near”** возвращает документы, которые находятся в указанном диапазоне не менее 1000 метров от указанной точки GeoJSON и не более 5000 метров от нее, и, следовательно, мы получаем только выходные данные Tidal Park. Подобно $near, он может поддерживать $ nearSphere, $ geoWithin, $ geoIntersects, $ geoNear и т.д.,



5. Текстовый индекс: MongoDB поддерживает операции запроса, которые выполняют текстовый поиск содержимого строки. Текстовый индекс позволяет нам находить содержимое строки в указанной коллекции. Он может включать любое поле, содержащее строковое содержимое или массив строковых элементов. Коллекция может содержать не более одного текстового индекса. Вам разрешено использовать текстовый индекс в составном индексе.

Синтаксис:

db.<коллекция>.createIndex({ <поле>: «текст»})

Мы также можем указать точные фразы для поиска, заключив поисковые термины в двойные кавычки

db.<имя коллекции>.find({ $text: { $search: «\”<Точный поисковый запрос>\”” } })

Поскольку здесь результаты поиска заключены в двойные кавычки, они содержат только точные искомые данные.

В случае, если мы хотим исключить несколько текстов из нашего поискового запроса, мы можем сделать следующее

*db.<имя коллекции>.find({ $text: { $search: «***<условия поиска>** - **<необязательные условия поиска>**” } })*

Добавление символа – делает текст поиска игнорируемым, а остальной текст учитывается.

При текстовом поиске результаты доступны в несортированном порядке. Чтобы сделать их доступными в отсортированном порядке оценки

релевантности, необходимо поле $meta textScore и выполнить сортировку по нему. Пример:

```
db.singers.find(
  { $text: { $search: «Annisten» } },
  { score: { $meta: «textScore» } }
).sort( { score: { $meta: «textScore» } } )
```

Пример:

В коллекции аксессуаров мы создаем текстовый индекс:

```
db.accessories.createIndex({name: «text», description: «text»})
```

```
> db.accessories.find()
{ "_id" : 1, "name" : "keyboard", "description" : "Input device" }
{ "_id" : 2, "name" : "Mouse", "description" : "Input device" }
{ "_id" : 3, "name" : "Printer", "description" : "Output device" }
> db.accessories.createIndex( { name: "text", description: "text" } )
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Теперь мы отобразим те документы, которые содержат строку «Input”:

```
db.accessories.find({$text:{$search: «Input»}})
```

6. Хэш-индекс: Чтобы поддерживать записи с хэшами значений индексируемого поля (в основном поля _id во всех коллекциях), мы используем хэш-индекс. Этот тип индекса в основном требуется для равномерного распределения данных посредством сегментирования. Хэшированные ключи полезны для разделения данных по сегментированному кластеру.

Синтаксис:

db.<коллекция>.createIndex({ _id: «хэшированный” })

Начиная с версии 4.4, применяется составной хэшированный индекс

7. Подстановочный индекс: MongoDB поддерживает создание индексов либо для поля, либо для набора полей, и если упоминается набор полей, это называется подстановочным индексом. Как правило, подстановочный индекс не включает поле _id, но если вы хотите включить поле _id в подстановочный индекс, вы должны определить его явно. MongoDB позволяет создавать несколько подстановочных индексов в данной коллекции. Подстановочные индексы поддерживают запросы к неизвестным или произвольным полям.

Синтаксис:

Для создания подстановочного индекса в указанном поле:

*db.<коллекция>.createIndex({ «поле.$**”: 1 })*

Создать подстановочный знак для всего поля:

*база данных.<коллекция>.createIndex({ «$**”:1 })*

Для создания подстановочного индекса для нескольких заданных полей:

база данных.<коллекция>.createIndex(
*{ «$**”:1 },*
{«wildcardProjection”:
{”field1»: 1, «field2”: 2}

})

Пример:

В коллекции книг мы создаем подстановочный индекс:

```
> db.book.find().pretty()
{
        "_id" : 1,
        "title" : "mongodb with java",
        "categories" : [
                "mongodb"
        ],
        "authors" : [
                1,
                2
        ],
        "authorTags" : {
                "inclusions" : [
                        "RDBMS",
                        "NoSQL"
                ],
                "usedin" : [
                        "SecurityApplications",
                        "Multipurpose"
                ]
        }
}
```

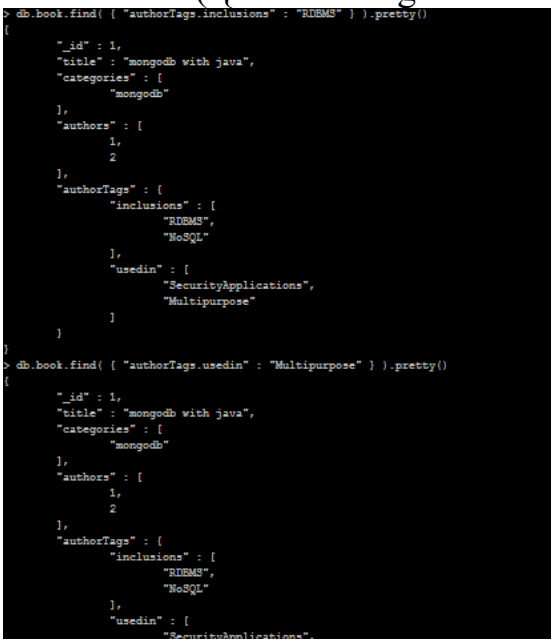
Давайте создадим индекс для поля «authorTags”

db.book.createIndex({ «authorTags.$**» : 1 })

Поскольку «индекс” создается на основе набора полей, мы можем легко выполнить запрос следующим образом

db.book.find({ «authorTags.inclusions» : «RDBMS» })

db.book.find({ «authorTags.usedin» : «Multipurpose» })



Задание:

Выполнить запросы к базе данных с индексами и без. Сравнить время выполнения запросов. Сделать выводе о проведенном эксперименте.

Технология выполнения работы:

1. Скачайте json файл с данными о ресторанах по ссылке mongodb-json-files/datasets/restaurant.json at master · ozlerhakan/mongodb-json-files (github.com)
2. Установите и откройте MongoDB Compass. Создайте базу данных с произвольным именем, а в ней коллекцию restaurants
3. В коллекцию restaurants с помощью импортируйте данные о ресторанах из скачанного файла.
4. В интерфейсе MongoDB Compass откройте консоль и выполните запрос на получение всех документов с любым фильтром. Проверьте, что все работает корректно. Засеките время выполнения запроса.
5. Создайте индекс, который ускорит выполнение запроса.
6. Выполните запрос с индексом. Засеките время.
7. Создайте отчет с выводами об индексах и времени выполнения запроса.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Выполнены запросы;
- Проведено сравнение времени выполнения запроса с индексом и без индекса;
- Создан итоговый отчет

Контрольные вопросы:

1. Что представляют собой индексы в базах данных?
2. Какие преимущества предоставляют индексы в управлении базами данных?
3. Какие типы индексов существуют?
4. Какие оптимизации происходят при использовании индексов в запросах?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.

4. 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.

Практическое занятие № 4

Тема практического занятия: Планирование БД и коллекций, моделирование данных

Цель практического занятия: Освоить методы планирования бд и коллекций

В результате выполнения данной работы обучающийся должен уметь:

- выполнять проектирование нереляционных баз данных;
- создавать диаграммы БД и коллекций;
- создавать БД и коллекции любой сложности

знать:

- принципы проектирования No-Sql баз данных;
- виды диаграмм, используемых при проектировании;
- методы планирования БД

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

В основе моделирования баз данных лежит идея создания структуры базы данных, которая определяет, как можно получить доступ к хранимой информации, классифицировать ее и манипулировать ею. Это сама основа проектирования базы данных, а конкретная модель данных, используемая при проектировании, определяет диаграмму базы данных и общие усилия по разработке. Читайте дальше, чтобы узнать, почему моделирование является инженерным императивом, а также некоторые из наиболее популярных методов моделирования данных.

Моделирование базы данных

По сути, база данных должна быть простой в использовании и должна поддерживать целостность данных в безопасном режиме. Мощная модель базы данных также позволит использовать различные способы управления, контроля и организации хранимой информации для эффективного

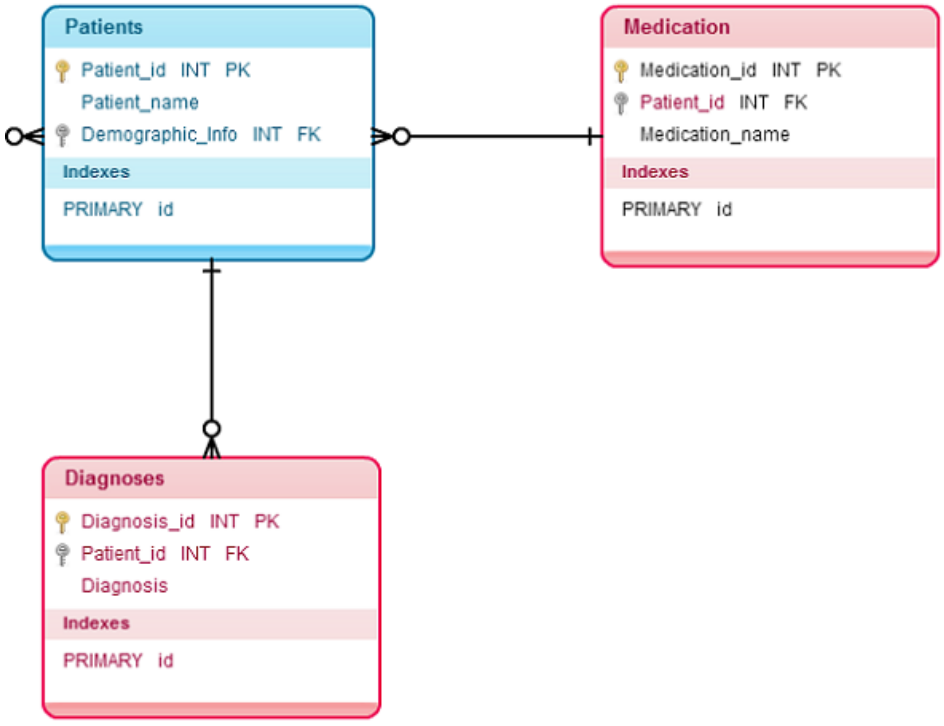
выполнения множества ключевых задач. На этапе проектирования диаграммы базы данных будут содержать необходимую документацию о ссылках на данные, которые облегчают функциональность базы данных.

Типы методов моделирования баз данных

Ниже приведен список наиболее распространенных методов моделирования баз данных. Обратите внимание, что в зависимости от типа данных и потребностей конечного пользователя при доступе к базе данных можно использовать несколько моделей для создания более сложного дизайна базы данных. Разумеется, в обоих сценариях для установления и поддержания высоких эксплуатационных стандартов потребуется создание диаграмм базы данных. К счастью, готовые под ключ диагностические и дизайнерские инструменты, такие как Creately, могут сделать это усилие легким ветерком.

Из приведенных ниже моделей реляционная модель является наиболее часто используемой моделью для большинства проектов баз данных. Но в некоторых особых случаях другие модели могут быть более выгодными. К счастью, Creately поддерживает все модели.

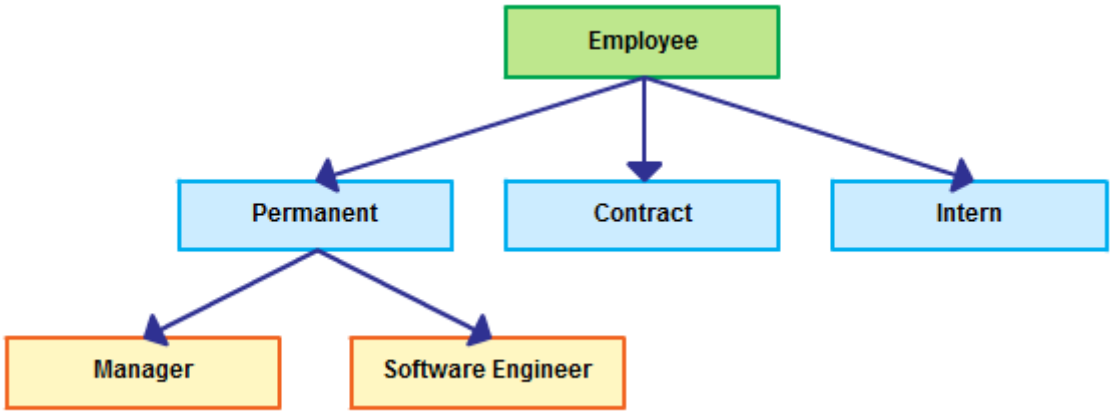
Реляционная модель: Основанная на математической теории, эта модель базы данных выводит хранение и поиск информации на новый уровень, так как она предлагает способ поиска и понимания различных взаимосвязей между данными. Рассматривая, как различные переменные могут изменить соотношение между данными, можно получить новые перспективы по мере изменения представления информации, сосредоточившись на различных атрибутах или областях. Эти модели часто можно найти в системах бронирования авиабилетов или в базах данных банков.



Метод реляционного проектирования, наиболее популярный метод проектирования баз данных

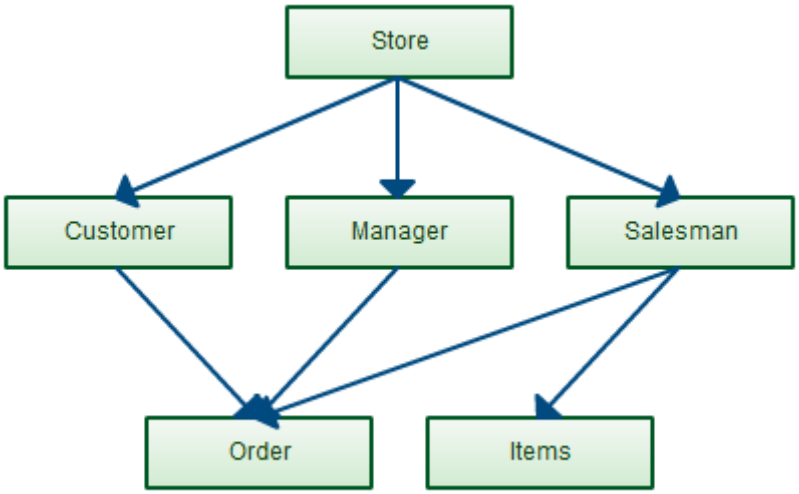
Модель графика: модель графика – еще одна модель, которая набирает популярность. Эти базы данных созданы на основе теории графов и используют узлы и ребра для представления данных. Структура несколько похожа на объектно-ориентированные приложения. Базы данных графиков, как правило, легче масштабировать и обычно быстрее работают с ассоциативными наборами данных.

Иерархическая модель: Как и общая организационная схема, используемая для организации компаний, эта модель базы данных имеет тот же древовидный вид и часто используется для структурирования XML документов. При рассмотрении эффективности данных это идеальная модель, в которой данные содержат вложенную и отсортированную информацию, но она может быть неэффективной, когда данные не имеют восходящей связи с основной точкой данных или субъектом. Данная модель хорошо работает для системы управления информацией сотрудников в компании, которая стремится ограничить или назначить использование оборудования определенным лицам и/или отделам.



Иерархический метод, самая первая модель проектирования базы данных.

Сетевая модель: Используя записи и наборы данных, эта модель использует подход «от одного к другому” к записям данных. Несколько ветвей выделяются для структур нижнего уровня и ветвей, которые затем соединяются несколькими узлами, представляющими собой структуры верхнего уровня внутри информации. Этот метод моделирования базы данных обеспечивает эффективный способ получения информации и организации данных таким образом, чтобы их можно было рассматривать несколькими способами, обеспечивая средство для увеличения производительности бизнеса и времени отклика. Это жизнеспособная модель для планирования дорожных, железнодорожных или коммунальных сетей.



Сетевая модель, в которой узел может иметь несколько родительских узлов

Размерная модель: Это является адаптацией реляционной модели и часто используется в сочетании с ней путем добавления «размерности" фактов к точкам данных. Эти факты могут быть использованы в качестве измерительных палочек для других данных для определения того, как размер группы или время группы повлияли на определенные результаты. Это может помочь бизнесу принимать более эффективные стратегические решения и помочь им узнать свою целевую аудиторию. Эти модели могут быть полезны организациям с анализом продаж и прибыли.

Объектная реляционная модель: Эти модели создали совершенно новый тип базы данных, который сочетает в себе дизайн базы данных с прикладной программой для решения конкретных технических задач, используя при этом лучшее из обоих миров. На сегодняшний день базы данных объектов все еще нуждаются в доработке для достижения большей стандартизации. Применение этой модели в реальном мире часто включает в себя технические или научные области, такие как инженерия и молекулярная биология.

В этой практической мы рассмотрим, как смоделировать структуру, содержащую списки рассылок и данные о людях, которые входят в эти списки.

Ниже представлены требования:

- Человек может иметь один или более e-mail адресов;
- Человек может состоять в любом количестве списков рассылок;
- Человек может выбрать любое название для любого списка рассылки, в котором состоит.

Стратегия «без встраиваний»

Давайте посмотрим, как будет выглядеть наша модель данных, если никакие данные никуда не встраивать. У нас есть подписчики People с именами и паролями:

```
{
    _id: PERSON_ID,
```

```
    name: «Василий Пупкин»,
    pw: «Хешированный пароль»
}
```

У нас есть коллекция адресов **Adresses**, в которой каждый документ содержит e-mail адрес и привязку к конкретному подписчику:

```
{
    _id: ADDRESS_ID,
    person: PERSON_ID,
    address: «vpupkin@gmail.com»
}
```

У нас есть группы **Groups**, каждая из которых содержит только идентификатор группы (она, конечно же, может содержать и другие данные, но мы специально опустим этот момент, дабы сконцентрироваться на подписках)

```
{
    _id: GROUP_ID
}
```

И наконец, мы имеем коллекцию подписок Memberships. Каждая Подписка объединяет людей в Группы, кроме этого, содержит название, которое человек выбрал для данной Группы, и ссылку на e-mail адрес, который он хочет использовать для получения рассылки в данной Группе:

```
{
    _id: MEMBERSHIP_ID,
    person: PERSON_ID,
    group: GROUP_ID,
    address: ADDRESS_ID,
    group_name: «Семья»
}
```

Такая модель данных понятна, легка в разработке и проста в обслуживании. Мы создали модель, которую удобно использовать в реляционной базе данных. При этом мы совсем не приняли во внимание документо-ориентированный подход MongoDB. Давайте рассмотрим, что мы будем делать, чтобы получить, например, e-mail адреса всех членов одной Группы, имея один известный e-mail адрес и название этой Группы:
В коллекции **Addresses** по известному e-mail найдем PERSON_ID;

1) В коллекции **Memberships** по полученному PERSON_ID и известному названию Группы найдем GROUP_ID;

2) Опять же в коллекции **Memberships** по полученному GROUP_ID найдем список Подписок данной Группы;
3) И наконец из коллекции **Addresses** по ADDRESS_ID, пройдя по каждой Подписке из полученного списка, получим список e-mail адресов.

Стратегия «все встроено»

Теперь рассмотрим случай, когда все данные встроены в один документ. Для этого мы возьмем все Подписки Группы и встроим их в модель Группы. Плюсом в каждую Подписку встроим данные об Подписчике и его e-mail адресах:

```
{
    _id: GROUP_ID,
    memberships: [{
        address: «vpupkin@gmail.com»,
        name: «Василий Пупкин»,
        pw: «Хешированный пароль»,
        person_addresses: [«vpupkin@gmail.com», «vpupkin@mail.ru», ...],
        group_name: «Семья»
    }, ...]
}
```

Смысл встраивать все связные данные в один документ заключается в том, что теперь некоторые запросы к данным делать намного проще. Запрос из предыдущей части статьи становится совсем простым (помните, нам нужно, имея один известный e-mail адрес и название Группы, узнать e-mail адреса остальных членов этой Группы):

1) В коллекции Groups найдем Группу, содержащую Подписку, в которой group_name совпадает с известным нам названием Группы и массив person_addresses содержит известный нам e-mail;
2) Разберем полученный документ для извлечения остальных e-mail адресов.

Гораздо проще. Но что будет, если Подписчик захочет поменять имя или пароль? Нам придется менять его имя или пароль в каждой встроенной Подписке каждой Группы, в которой состоит этот Подписчик. Это также касается добавления нового или удаление существующего e-mail адреса из массива person_addresses. Такие моменты говорят нам об определенном характере данной модели: она хорошо подходит для специфичных запросов (потому что все необходимые данные уже внутри, типа pre-join), но может стать кошмаров в долгосрочной перспективе в плане сопровождения.

Стратегия «частичного встраивания»

Подход, который я чаще всего рекомендую, это начать думать о модели данных без встраивания. После того, как у вас есть черновая модель, можно начинать выделять случаи, когда встраивание имеет смысл. Как правило, это отношения один-ко-многим.

Например, несолько e-mail адресов из коллекции Addresses принадлежат одному Подписчику (они также участвуют в модели Подписки) и обычно меняются не так часто. Поэтому мы объединим их в массив и добавим в нашу модель Подписчика, сделав её чуточку схожей с ментальной моделью.

Каждая Подписка связана с конкретным Подписчиком и конкретной Группой, поэтому можно встроить Подписки как в модель Подписчика, так и в модель Группы. В подобных случаях важно думать и о модели доступа к данным и о размере встраиваемых данных. Мы ожидаем, что люди вряд ли подпишутся на рассылку более чем из 1000 разных групп, и отдельно взятая группа в свою очередь также вряд ли наберет более 1000 подписчиков. В данном случае, цифры нам ничего полезного не говорят. Однако, наша модель доступа к данным, напротив, говорит нам, что при выводе на экран необходимо видеть все подписки конкретного человека. Для упрощения запроса мы встроим Подписки в модель Подписчика. Преимущество ещё и в том, что список e-mail адресов Подписчика находятся в модели Подписчика, а в Подписке используется один из адресов этого списка, и если нам нужно изменить или удалить e-mail адрес, это можно сделать в одном месте.

Теперь наша модель данных выглядит так:

```
{
    _id: PERSON_ID,
    name: «Василий Пупкин»,
    pw: «Хешированный пароль»,
    addresses: [«vpupkin@gmail.com», «vpupkin@mail.ru», ...],
    memberships: [{
        address: «vpupkin@gmail.com»,
        group_name: «Семья»,
        group: GROUP_ID
    }, ...]
}
```

1) Это модель Подписчика, кроме неё есть ещё модель Группы, которая идентична той, что рассмотрна в описании стратегии «без встраивания». Запрос, который мы обсуждали выше, теперь будет выглядеть так: В коллекции People найдем Подписчика с искомым e-mail адресом, среди Подписок которого есть Подписка с искомым названием;

2) Используя GROUP_ID найденной Подписки, найдем в коллекции People других Подписчиков этой Группы и возьмем их e-mail адреса непосредственно из Подписки.

Задание:

Спроектировать БД в MongoDB, спроектировать коллекции. Создать диаграммы спроектированных объектов. Создать базу в Mongo DB и заполнить данными. Проверить работоспособность созданной базы.

Технология выполнения работы:

1. Спроектируйте базу данных mongo для хранения информации о туристических местах России. Спроектируйте коллекции. Учтите необходимые параметры места.
2. Результаты проектирования п.1. представьте с помощью диаграмм. Создайте UML диаграмму любой коллекции
3. Создайте спроектированную базу в mongo db и заполните данными.
4. В отчете прикрепите схему, диаграммы и пример каждой коллекции из бд.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Выполнено проектирование БД и коллекций
- Созданы диаграммы
- Создан отчет со скриншотами проделанной работы
- Даны ответы на контрольные вопросы

Контрольные вопросы:

1. Виды диаграмм, показывающих схему БД?
2. Какие есть модели данных?
3. Какая модель данных используется в Mongo DB?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 5

Тема практического занятия: Создание кластера БД. Подключение, резервное копирование и восстановление.

Цель практического занятия: научиться создавать отказоустойчивый кластер БД и освоить методы работы с ним.

В результате выполнения данной работы обучающийся должен уметь:

- создавать кластер Greenplum;
- выполнять резервное копирование БД;
- восстанавливать БД

знать:

- особенности распределенных СУБД;
- принципы работы распределенных СУБД;
- как работать с утилитами pg_dump и pg_restore

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Greenplum - это распределенная система управления базами данных (СУБД), разработанная для обработки больших объемов данных и аналитики. Она основана на базе PostgreSQL и предназначена для работы с массивными наборами данных и выполнения сложных аналитических запросов. Greenplum использует архитектуру Shared Nothing, где данные разделены между несколькими серверами без общей памяти или диска. Кластер Greenplum состоит из мастер-сервера и нескольких сегментов (segment), каждый из которых состоит из одного или нескольких сегментных экземпляров базы данных. Мастер-сервер координирует работу в кластере, а сегменты выполняют фактическую обработку данных. Основной принцип Greenplum состоит в разделении и фрагментировании данных, что позволяет обеспечить параллельную обработку запросов. Данные разбиваются на сегменты и распределяются между ними в зависимости от ключей или хэш-функций. Каждый сегмент обрабатывает только свою часть данных, что позволяет

значительно ускорить выполнение запросов. Greenplum также обладает эффективными механизмами сжатия данных, индексации и оптимизации запросов. Он поддерживает SQL-совместимость с PostgreSQL, что облегчает портирование и использование существующих приложений. Основные возможности Greenplum включают в себя параллельное выполнение запросов, масштабируемость, высокую производительность и поддержку неструктурированных данных. Он применяется в различных областях, включая аналитику больших данных, бизнес-интеллект, маркетинговые исследования и многое другое. Greenplum - это мощная и гибкая система управления базами данных, которая позволяет эффективно обрабатывать большие объемы данных и выполнять сложные аналитические запросы. Его расширяемая архитектура и широкий набор функциональных возможностей делает его одним из привлекательных выборов для различных предприятий и организаций, работающих с большими данными.

Задание: Создайте кластер БД. Проведите базовую настройку кластера. Создайте базу данных и выполните задания по работе с кластером. Выполните резервное копирование и восстановление базы данных.

Технология выполнения работы:

1. Установите Greenplum на свою локальную машину или используйте доступ к удаленному серверу с Greenplum.
2. Создайте новый кластер базы данных PostgreSQL с помощью Greenplum.
3. Настройте параметры кластера, такие как количество сегментов (segment), мощность обработки данных (processing power), размер данных.
4. Подключитесь к кластеру с помощью клиента PostgreSQL,, через графический интерфейс, такой как pgAdmin.
5. Создайте новую базу данных в кластере и выполните некоторые базовые запросы для проверки подключения и функциональности.
6. Проведите резервное копирование данных в кластере с помощью Greenplum. Удостоверьтесь, что копия данных создана успешно и может быть использована для восстановления.
7. Случайно удалите несколько записей из базы данных и восстановите их из резервной копии, используя механизм восстановления данных Greenplu
8. Проверьте, что восстановленные данные соответствуют оригинальным данным, и подтвердите успешность процесса восстановления.
9. Завершите выполнение задания, предоставив отчет о выполненных шагах, и описывающим опыт создания кластера, особые моменты подключения, резервного копирования и восстановления данных.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Создан кластер Greenplum;
- Проведена настрйка кластера;
- Создана БД;
- Произведено резервное копирование и восстановление;
- Предоставлен отчет о работе

Контрольные вопросы:

1. Что такое реляционные БД?
2. Какие плюсы имеют реляционные БД?
3. Как создать и настроить кластер Greenplum?
4. Как создать бд, седлать дамп и восстановление?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 6

Тема практического занятия: Шардирование

Цель практического занятия: ознакомить студентов с концепцией и практикой шардирования, которое является важным аспектом в области управления данными и обеспечивает эффективное масштабирование баз данных.

В результате выполнения данной работы обучающийся должен уметь:

- проводить шардирование БД;
- оценивать качество шардирования;
- выполнять тонкую найтройку кластера при шпрдировании данных

знать:

- что такое шардирование;
- основные понятия, связанные с термином шардирования;
- этапы проведения шардирования

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Шардирование (sharding) - это метод горизонтального разделения данных по разным узлам или серверам в кластере для распределения нагрузки и увеличения производительности. Вместо того чтобы хранить все данные на одном узле, они разбиваются на фрагменты (шарды), которые хранятся на разных узлах. В кластере Greenplum шардирование реализуется с помощью декомпозиции таблицы на разные сегменты данных. Каждый сегмент может хранить данные отдельного шарда. Шарды можно сгруппировать в сегментные базы данных, которые содержат наборы шардов, и разместить эти сегментные базы данных на различных узлах в кластере. Для проведения шардирования в кластере Greenplum необходимо выполнить следующие шаги: 1. Создать сегментную базу данных (segment database). Сегментная база данных - это набор шардов на одном узле. Каждый узел может содержать несколько сегментных баз данных. 2. Создать шарды для таблицы, которую необходимо распределить. Шарды могут быть созданы для отдельных

столбцов или для комбинаций столбцов, в зависимости от требований производительности и доступности данных. 3. Распределить шарды по сегментным базам данных. Шарды можно равномерно распределить между сегментными базами данных, либо использовать эвристические алгоритмы для более оптимального размещения данных. 4. Создать таблицу с указанием результирующей схемы шардирования. При создании таблицы необходимо указать какие столбцы будут использованы для разделения данных на шарды, а также какие столбцы будут использованы для дальнейшего разделения на сегменты данных. После проведения шардирования данные будут храниться в распределенном формате по сегментным базам данных, что позволит более эффективно использовать ресурсы кластера и достигнуть лучшей производительности при выполнении запросов.

Задание: Создать базу данных**.** Провести шардирование БД Postgres в кластере Greenplum. Оценить качество проведенного шардирования. Проверить работоспособность кластера после шардирования. Подвести итоги.

Технология выполнения работы:

1. Установка и настройка кластера Greenplum - Скачайте и установите кластер Greenplum на свою локальную машину или на виртуальную машину. - Настройте конфигурационные файлы кластера для оптимальной работы с вашими ресурсами.
2. Создание и настройка базы данных - Создайте новую базу данных в кластере Greenplum. - Настройте параметры базы данных, такие как размеры сегментов, количество сегментов и конфигурационные параметры.
3. Создание сегментированной таблицы - Создайте новую таблицу в базе данных, которая будет сегментирована по заданной колонке. - Выберите подходящую колонку для шардирования, например, ID заказов.
4. Заполнение таблицы данными - Заполните созданную таблицу данными для тестирования. - Обратите внимание на изменение распределения данных после заполнения.
5. Выполнение запросов - Попробуйте выполнить различные запросы на сегментированную таблицу для проверки эффективности шардирования. - Сравните время выполнения запросов до и после шардирования.
6. Добавление нового сегмента - Добавьте новый сегмент в кластер Greenplum. - Проверьте, как изменится распределение данных и время выполнения запросов после добавления нового сегмента.
7. Оценка производительности - Оцените производительность вашей сегментированной таблицы и сравните ее с несегментированной таблицей. - Обратите внимание на изменение времени выполнения запросов и распределение данных.

8. В заключении работы опишите свои выводы о практическом применении шардирования базы данных PostgreSQL в кластере Greenplum.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Создан кластер Greenplum;
- Проведена настройка кластера;
- Создана БД и таблица;
- Проведен эксперимент с шардированием

Контрольные вопросы:

1. Что такое кластер баз данных?
2. Какие преимущества предоставляет использование кластера БД?
3. Какие типы резервного копирования данных вы знаете?
4. Какие этапы включает в себя процесс создания кластера баз данных?
5. Как происходит подключение к кластеру баз данных?
6. Что такое точка восстановления базы данных?
7. Какие методы восстановления базы данных существуют?
8. Что такое шардинг в контексте кластера баз данных?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 7

Тема практического занятия: Создание хранилища данных DWH. Изменение модели данных

Цель практического занятия: практическое овладение основами и процессами создания и управления хранилищем данных

В результате выполнения данной работы обучающийся должен уметь:

- создавать хранилище данных;
- управлять хранилищем данных;
- проектировать хранилища данных DWH для конкретных кейсов;
- проводить аудит хранилища данных

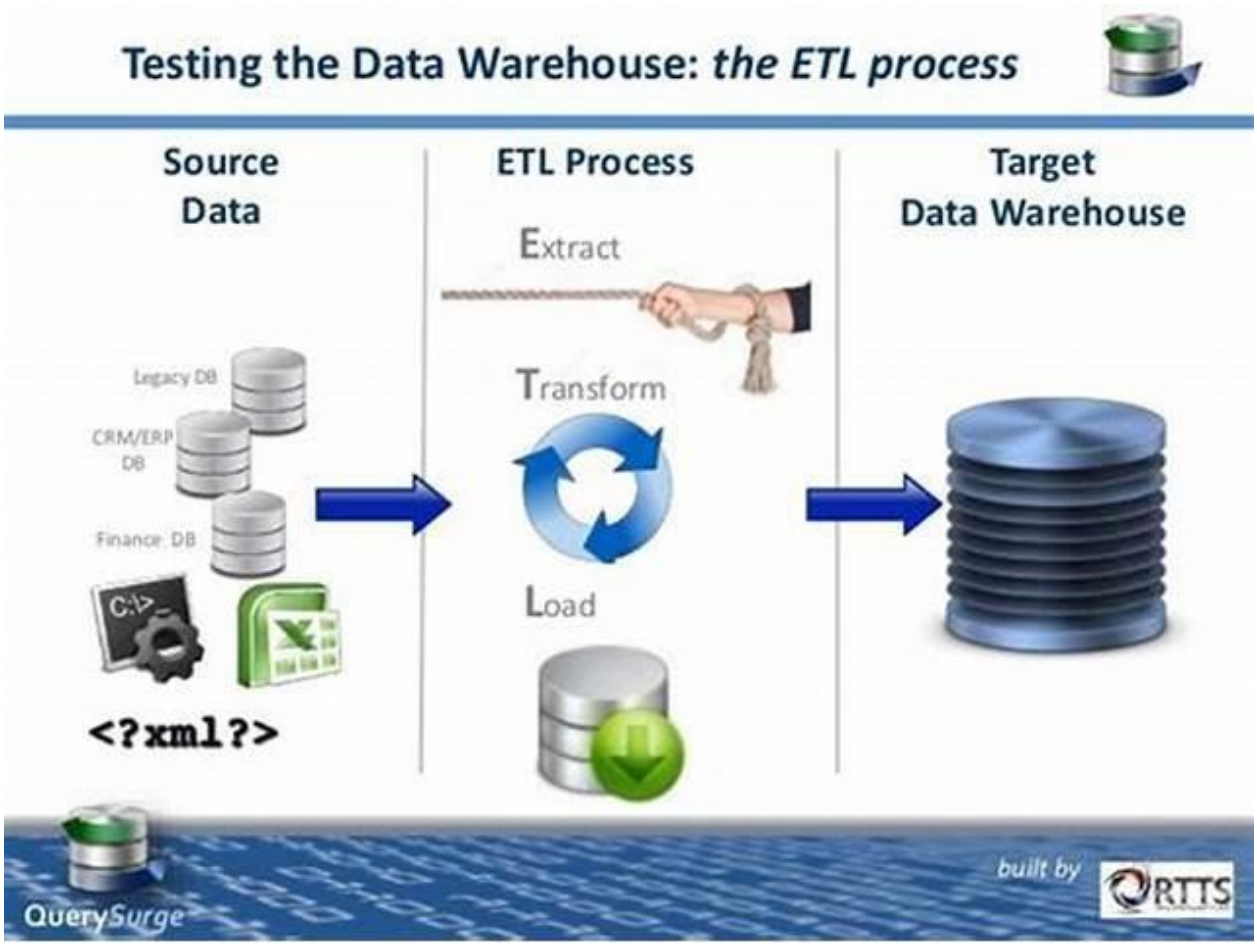
знать:

- что такое хранилище данных;
- какие процессы связаны с DWH;
- основные понятия, связанные с термином хранилища данных DWH

Общие теоретические сведения

Data warehouse — склад всех нужных и важных для принятия решений данных компании. Но есть же всякие базы данных внутри фирмы, разве они не DWH? Например, СУБД с клиентами, складскими запасами или покупками. Где разница между обычной базой данных и DWH? Разница вот в чем:

1. **Типы хранимых данных**. Обычные СУБД хранят данные строго для определенных подсистем. База данных склада хранит складские запасы и ничего более. База данных кадровиков хранит данные по персоналу, но не товары или сделки. DWH, как правило, хранит информацию разных подразделений — там найдутся данные и по товарам, и по персоналу, и по сделкам.
2. **Объемы данных**. Обычная БД, которая ведется в рамках стандартной деятельности компании, содержит только актуальную информацию, нужную в данный момент для функционирования определенной системы. В DWH пишутся не столько копии актуальных состояний, сколько исторические данные и агрегированные значения. Например, состояние запасов разных категорий товаров на конец смены за последние пять лет. Иногда в DWH пишутся и более крупные пачки данных, если они имеют критическое значение для бизнеса — допустим, полные данные по продажам и сделкам. То есть, по сути, это копия СУБД отдела продаж.
3. **Место в рабочих процессах**. Информация обычно сразу попадает в рабочие базы данных, а уже оттуда некоторые записи переползают в DWH. Склад данных, по сути, отражает состояние других БД и процессов в компании уже после того, как вносятся изменения в рабочих базах.



Для создания Data Warehouse (DWH) следуйте этим шагам:

1. Определите бизнес-требования: Понимание бизнес-процессов и данных, которые требуются для анализа, позволит определить необходимые источники данных и функциональные требования для DWH.
2. Выберите подход к DWH: Существуют два основных подхода к созданию DWH: топ-доун и боттом-ап. В топ-доун подходе вы сначала определяете общие потребности анализа и затем создаете DWH для удовлетворения этих потребностей. В боттом-ап подходе вы начинаете с небольших фрагментов данных и поэтапно создаете DWH на основе этих фрагментов.
3. Определите структуру и модель данных: Определите, какие данные необходимо хранить в DWH и как они будут организованы. Разработайте концептуальную, логическую и физическую модели данных для DWH.
4. Выберите инструменты и технологии: Определите, какие инструменты и технологии будут использоваться для создания DWH. Это может включать базы данных, ETL-инструменты, инструменты аналитики и др.
5. Создайте ETL-процессы: Разработайте процессы извлечения, преобразования и загрузки (ETL) для переноса данных из источников в DWH. Эти процессы должны быть автоматизированы и масштабируемыми.

6. Создайте хранилище данных: Используя выбранные инструменты и технологии, создайте физическое хранилище данных для DWH. Проектируйте и оптимизируйте схему базы данных для обеспечения быстрого доступа к данным.
7. Разработайте отчеты и аналитические инструменты: Разработайте отчеты и аналитические инструменты для доступа к данным в DWH. Это может включать создание дашбордов, OLAP-кубов, аналитических отчетов и т. д.
8. Тестируйте и оптимизируйте DWH: Осуществите тестирование и оптимизацию DWH для обеспечения высокой производительности и надежности. Выполняйте регулярное обслуживание и мониторинг DWH.
9. Деплойте DWH: Разверните DWH в production-среду и настройте его для доступа к остальным системам и пользователям. 1
10. Поддерживайте DWH: Проводите регулярное обслуживание и поддержку DWH, чтобы обеспечить его непрерывную работоспособность и актуальность данных.

Задание:

Спроектировать хранилище данных DWH. Создать хранилище данных DWH. Написать сопроводительную документацию к созданному хранилищу данных. Заполнить хранилище DWH данными.

Технология выполнения работы:

1. Спроектируйте схему DWH для хранения данных о поставках товаров в соответствии со справочником ОКПД2. Необходимо хранить данные о компаниях и товарах (включая все необходимые характеристики). Схема должна отображать источники информации, etl-процессы и хранилище данных.
2. Создайте хранилище данных. Требования к хранилищу – любая нереляционная или реляционная БД.
3. Создайте необходимые таблицы в БД.
4. Создайте отчет, в котором необходимо прикрепить схему DWH и ER – диаграмму БД.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Создано хранилище данных DWH;
- Спроектирована схема DWH;
- На схеме отображены все процессы

Контрольные вопросы:

1. Что такое DWH?
2. Что включает DWH?
3. Как создать DWH?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 8

Тема практического занятия: Проектирование ETL-процесса. Оценка качества данных

Цель практического занятия: ознакомление с основными принципами, методами и инструментами, связанными с проектированием процессов ETL (Extract, Transform, Load) и оценкой качества данных

В результате выполнения данной работы обучающийся должен уметь:

- создавать и отлаживать ETL-процессы;
- оценивать качество данных;
- писать программный код для всех стадий спроектированного ETL-процесса

знать:

- что такое ETL-процессы;
- критерии качества данных

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

ETL – аббревиатура от Extract, Transform, Load. Это системы корпоративного класса, которые применяются, чтобы привести к одним справочникам и загрузить в DWH и EPM данные из нескольких разных учетных систем.

Извлечение, преобразование и загрузка (ETL) улучшает бизнес-аналитику и аналитику, делая этот процесс более надежным, точным, подробным и эффективным.

Исторический контекст

ETL обеспечивает глубокий исторический контекст данных организации. Предприятие может объединить устаревшие данные с данными из новых платформ и приложений. Вы можете просматривать более старые наборы данных наряду с более свежей информацией, что позволяет получить долгосрочное представление о данных.

Консолидированное представление данных

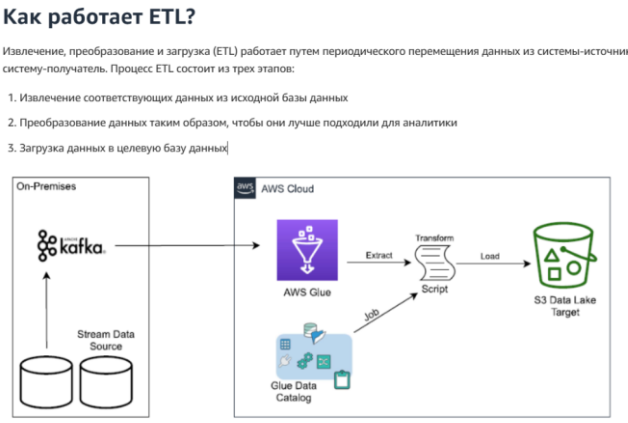
ETL обеспечивает консолидированное представление данных для углубленного анализа и отчетности. Управление многочисленными наборами данных требует времени и координации и может привести к неэффективности и задержкам. ETL объединяет базы данных и различные формы данных в единое, унифицированное представление. Процесс интеграции данных улучшает качество данных и экономит время, необходимое для перемещения, категоризации или стандартизации данных. Это облегчает анализ, визуализацию и осмысление больших массивов данных.

Точный анализ данных

ETL обеспечивает более точный анализ данных для соответствия нормативным и регулятивным стандартам. Вы можете интегрировать инструменты ETL с инструментами обеспечения качества данных для профилирования, аудита и очистки данных, обеспечивая их достоверность.

Автоматизация задач

ETL автоматизирует повторяющиеся задачи обработки данных для эффективного анализа. Инструменты ETL автоматизируют процесс миграции данных, и вы можете настроить их на периодическую интеграцию изменений данных или даже во время выполнения. В результате инженеры по обработке данных могут больше времени уделять инновациям и меньше – решению таких утомительных задач, как перемещение и форматирование данных.



1.2 Что такое извлечение данных?

При извлечении данных инструменты извлечения, преобразования и загрузки (ETL) извлекают или копируют необработанные данные из различных источников и сохраняют их в зоне хранения. Промежуточная среда (или целевая зона) – это промежуточная зона хранения для временного хранения извлеченных данных. Промежуточные среды часто являются

временными, то есть их содержимое стирается после завершения извлечения данных. Однако в промежуточной среде может также храниться архив данных для целей устранения неполадок.

Частота отправки данных из источника данных в целевое хранилище данных зависит от базового механизма сбора данных об изменениях. Извлечение данных обычно происходит одним из трех следующих способов.

1) **Уведомление об обновлении**

При уведомлении об обновлении система-источник уведомляет вас об изменениях в записи данных. Затем вы можете запустить процесс извлечения для этого изменения. Большинство баз данных и веб-приложений предоставляют механизмы обновления для поддержки этого метода интеграции данных.

2) **Инкрементное извлечение**

Некоторые источники данных не могут предоставлять уведомления об обновлении, но могут идентифицировать и извлекать данные, которые были изменены за определенный период времени. В этом случае система проверяет изменения через периодические промежутки времени, например, раз в неделю, раз в месяц или в конце кампании. Вам нужно извлекать только те данные, которые изменились.

3) **Полное извлечение**

Некоторые системы не могут определить изменения данных или выдать уведомление, поэтому единственным вариантом является перезагрузка всех данных. Этот метод извлечения требует, чтобы вы сохранили копию последнего извлечения, чтобы проверить, какие записи являются новыми. Поскольку этот подход предполагает большие объемы передачи данных, мы рекомендуем использовать его только для небольших таблиц.

1.3 Что такое преобразование данных?

При преобразовании данных инструменты извлечения, преобразования и загрузки (ETL) преобразуют и консолидируют исходные данные в зоне хранения, чтобы подготовить их для целевого хранилища данных. Этап преобразования данных может включать нижеследующие типы изменений данных.

1) **Базовое преобразование данных**

Базовые преобразования улучшают качество данных, удаляя ошибки, опустошая поля данных или упрощая их. Примеры этих преобразований приведены ниже.

2) **Чистка данных**

В процессе очистки данных удаляются ошибки и исходные данные приводятся к целевому формату. Например, вы можете сопоставить пустые поля данных с числом 0, сопоставить значение данных «Родитель» с «Р» или сопоставить «Дети» с «Д».

3) **Дедупликация данных**

Дедупликация при очистке данных выявляет и удаляет дублирующиеся записи.

4) **Пересмотр формата данных**

При пересмотре формата преобразуются данные, такие как наборы символов, единицы измерения и значения даты/времени, в согласованный формат. Например, у пищевой компании могут быть разные базы данных рецептов с ингредиентами, измеряемыми в килограммах и фунтах. ETL преобразует все данные в фунты.

5) **Расширенное преобразование данных**

При расширенных преобразованиях используются бизнес-правила для оптимизации данных для упрощения анализа. Примеры этих преобразований приведены ниже.

6) **Деривация**

При деривации применяются бизнес-правила к данным для вычисления новых значений на основе существующих. Например, можно преобразовать выручку в прибыль путем вычитания расходов или рассчитать общую стоимость покупки путем умножения цены каждого товара на количество заказанных товаров.

7) **Объединение**

При подготовке данных в процессе объединения связываются одни и те же данные из разных источников данных. Например, вы можете найти общую стоимость покупки одного товара, сложив стоимость покупки у разных поставщиков и сохранив в целевой системе только итоговую сумму.

8) **Разделение**

Вы можете разделить столбец или атрибут данных на несколько столбцов в целевой системе. Например, если источник данных сохраняет имя клиента как «Иванов Иван Иванович», вы можете разделить его на имя, отчество и фамилию.

9) **Суммирование**

В результате суммирования повышается качество данных за счет сокращения большого количества значений данных в меньший набор данных. Например, значения счета-фактуры по заказу клиента могут иметь множество различных небольших сумм. Вы можете обобщить данные за определенный период, сложив их, чтобы построить показатель пожизненной ценности клиента (CLV).

10) **Шифрование**

Вы можете защитить конфиденциальные данные для соблюдения законов о защите данных или конфиденциальности данных, добавив шифрование до того, как потоки данных будут переданы в целевую базу данных.

1.4 Что такое загрузка данных?

При загрузке данных инструменты извлечения, преобразования и загрузки (ETL) перемещают преобразованные данные из зоны хранения в целевое хранилище данных. Для большинства организаций, использующих ETL, этот процесс автоматизирован, четко определен, непрерывен и управляем пакетами. Далее описывается два метода загрузки данных.

1) **Полная загрузка**

При полной загрузке все данные из источника преобразуются и перемещаются в хранилище данных. Полная загрузка обычно происходит при первой загрузке данных из исходной системы в хранилище данных.

2) **Инкрементная загрузка**

При инкрементной загрузке инструмент ETL загружает дельту (или разницу) между целевой и исходной системами через регулярные промежутки времени. Он сохраняет дату последнего извлечения, так что загружаются только записи, добавленные после этой даты. Существует два способа реализации инкрементной загрузки.

3) **Потоковая инкрементная загрузка**

Если у вас небольшие объемы данных, вы можете передавать непрерывные изменения по конвейерам данных в целевое хранилище данных. Когда скорость данных возрастает до миллионов событий в секунду, можно использовать обработку потока событий для мониторинга и обработки потоков данных, чтобы принимать более своевременные решения.

4) **Пакетная инкрементная загрузка**

Если у вас большие объемы данных, вы можете периодически собирать изменения данных загрузки в пакеты. В течение этого заданного периода времени никакие действия не могут происходить ни в исходной, ни в целевой системе, поскольку данные синхронизируются.

Задание:

Спроектировать ETL-процессы. Для каждой стадии ETL-процесса написать программный код. Отладить процессы. Проверить работоспособность системы. Оценить качество данных.

Технология выполнения работы:

1. Используя схему из предыдущей лабораторной работы проведите парсинг данных о компаниях и товарах. (Extract)
2. Преобразуйте данные к единому формату, проведите чистку и валидацию данных (Transform)
3. Загрузите данные в БД из предыдущей лабораторной работы
4. Оцените данные, исходя из следующих характеристик

Название	Краткое описание
Стандартность	Форматы данных и значений соответствуют принятым
Полнота	Набор данных содержит все необходимые элементы
Консистентность	Отсутствие конфликтов между возможными значениями
Уникальность	Отсутствие дублирующихся значений
Актуальность	Временной промежуток соответствует задаче
Валидность	Данные содержат те значения, которые должны

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Созданы и отлажены ETL-процессы;
- Данные успешно загружены в БД;
- Проведена оценка собранных данных

Контрольные вопросы:

1. Что означает аббревиатура ETL?
2. Какие задачи выполняет процесс извлечения (Extract) в ETL?
3. Что включает в себя этап преобразования (Transform) в процессе ETL?
4. Какие виды загрузки (Load) данных существуют в ETL?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 9

Тема практического занятия: Проектирование Data Lake

Цель практического занятия: ознакомление с основными принципами хранения и обработки данных в Data Lake

В результате выполнения данной работы обучающийся должен уметь:

- проектировать озера данных для различных кейсов;
- поддерживать озеро данных;
- проводить анализ озера данных

знать:

- основные понятия Data Lake;
- плюсы и минусы подхода озера данных

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

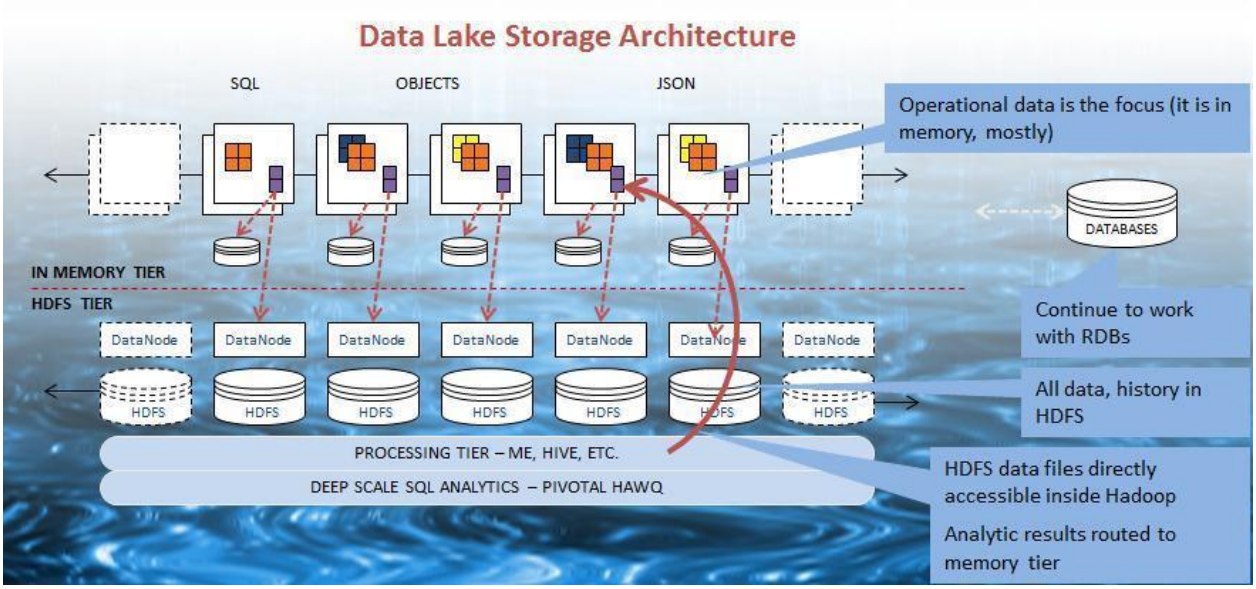
Data Lake (**Озеро данных**) — это метод хранения данных системой или репозиторием в натуральном (**RAW**) формате, который предполагает одновременное хранение данных в различных схемах и форматах. Обычно используется **blob-объект** (**binary large object**) или файл. Идея озера данных в том, чтобы иметь логически определенное, единое хранилище всех данных в организации (**enterprise data**) начиная от сырых, необработанных исходных данных (**RAW data**) до предварительно обработанных (**transformed**) данных, которые используются для различных задач: отчеты, визуализация, аналитика и машинное обучение.

Data Lake (**озеро данных**) включает структурированные данные из реляционных баз данных (строки и колонки), полуструктурированные данные (CSV, лог файлы, XML, JSON), неструктурированные данные (почтовые сообщения, документы, pdf) и даже бинарные данные (видео, аудио, графические файлы).

Data Lake (**озеро данных**), кроме методов хранения и описания данных, предполагает определение источников и методов пополнения данных. При этом используются следующие **термины**:

- источники — **sources**;
- настройки каналов — **pipelines**;
- регулярность обновлений — **schedulers;**
- владельцы — **custodians;**
- время хранения — **retention time;**
- метаданные — другие «данные о данных».

Data Lake (**озеро данных**) может использовать единый репозиторий в качестве хранилища данных (HDFS, **EDW**, **IMDG**, **Cloud** и т.д.) либо использовать модульную концепцию источников хранения данных для разных требований по безопасности, скорости, доступности при соблюдении условий хранения данных: неизменяемые **RAW** данные, согласованное время хранения (**retention time**), доступность.



Спроектировать Data Lake можно следующим образом:

1. Определить бизнес-цели: Определите, какие данные вам нужны, какие задачи вы хотите решить и какие вопросы вы хотите ответить с помощью Data Lake. Это поможет определить критические данные, которые нужно собрать.
2. Определить источники данных: Определите, откуда вы будете получать данные. Это могут быть базы данных, сторонние сервисы, приложения, устройства IoT и другие источники данных.
3. Обработка данных: Разработайте процедуры для сбора, очистки, трансформации и хранения данных в Data Lake. Решите, какие данные нужно преобразовать и какие структуры данных использовать.
4. Архитектура Data Lake: Определите архитектуру Data Lake. Вам нужно решить, какие технологии использовать, какая будет структура хранения данных, какие правила доступа и безопасности применять и т. д.

5. Выбор платформы: Выберите платформу или инструменты для создания и управления Data Lake. Возможно, вам понадобится комбинация открытых и коммерческих решений в зависимости от ваших потребностей.
6. Развитие и масштабирование: Проектирование Data Lake - это длительный процесс, который может потребовать последующего изменения и развития. Планируйте мероприятия по масштабированию и развитию вашего Data Lake в соответствии с бизнес-потребностями.
7. Обеспечение безопасности: Убедитесь, что ваши данные защищены при хранении и обработке в Data Lake. Разработайте стратегии обеспечения безопасности, включая контроль доступа, аудит, шифрование и мониторинг.
8. Управление метаданными: Разработайте систему управления метаданными, чтобы легко находить, оценивать и использовать данные в Data Lake. Метаданные помогут вам понять структуру данных и отношения между ними.
9. Аналитика данных: Разработайте процессы и инструменты для анализа данных в Data Lake. Решите, какие типы анализа данных вы хотите проводить и какие техники и модели использовать для получения ценной информации из данных.

Задание:

Спроектировать озеро данных по заданному кейсу. Создать озеро данных и заполнить данными. Разработать документацию к созданному озеру данных. Подвести итоги и создать итоговый отчет.

Технология выполнения работы:

1. Выберите кейс для проектирования:
- Фабрики на территории РФ
- Данные о недвижимости
- Данные о спортивных командах
- Курсы валют
- Новостные ленты
- Достопримечательности городов
- Данные о фильмах
- Данные о пользователях социальной сети
- Записи с камер видеонаблюдения на дорогах
2. Спроектируйте схему хранения данных.
3. Разработайте систему пополнения данных в вашем хранилище.
4. Разработайте схему метаданных, она должна отражать структуру данных.
5. Разработайте инструкцию для получения данных из хранилища.
6. Занесите полученные схемы в отчет. Также опишите плюсы и минусы вашего озера данных.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Спроектировано озеро данных;
- Описаны плюсы и минусы озера данных;
- Предоставлен отчет со всеми схемами

Контрольные вопросы:

1. Что такое Data Lake?
2. Как спроектировать Data Lake?
3. Назовите основные процессы, связанные с озером данных.

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 10

Тема практического занятия: Управление распределенной файловой системой Hadoop (HDFS)

Цель практического занятия: ознакомление с основными понятиями распределенных файловых систем и их применением в больших данных

В результате выполнения данной работы обучающийся должен уметь:

- администрировать кластер Hadoop;
- создавать кластер Hadoop с помощью Docker;
- работать с Docker и Docker Compose;
- выполнять основные команды для работы с распределенной файловой системой Hadoop

знать:

- что такое Hadoop, Hadoop HDFS;
- команды для управления кластером

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Hadoop – это свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов. Эта основополагающая технология хранения и обработки больших данных (Big Data) является проектом верхнего уровня фонда Apache Software Foundation.

Изначально проект разработан на Java в рамках вычислительной парадигмы MapReduce, когда приложение разделяется на большое количество одинаковых элементарных заданий, которые выполняются на распределенных компьютерах (узлах) кластера и сводятся в единый результат.

Проект состоит из основных 4-х модулей:

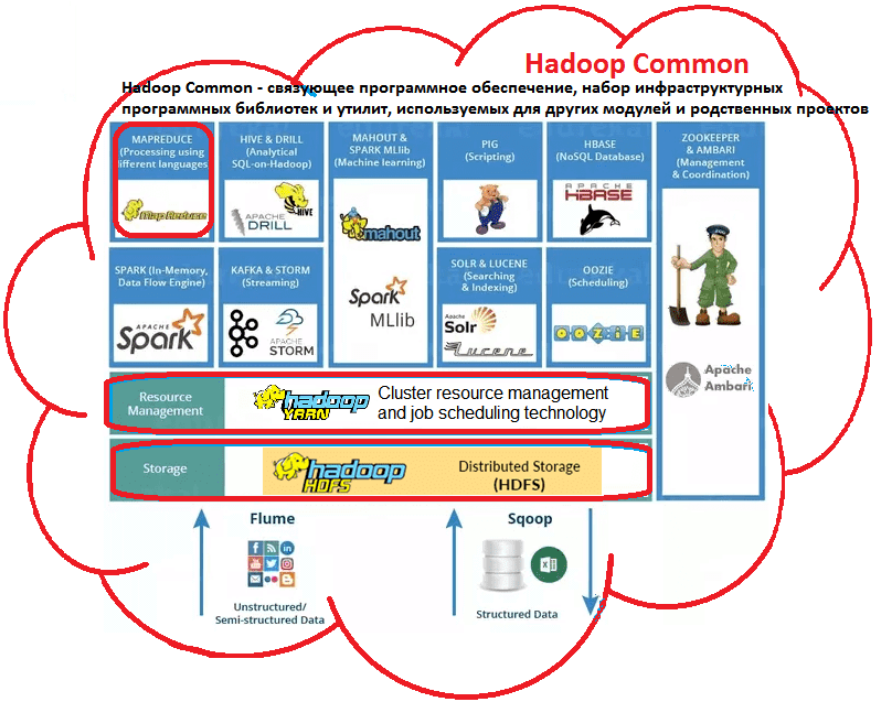
1. **Hadoop Common** – набор инфраструктурных программных библиотек и утилит, которые используются в других решениях и родственных

проектах, в частности, для управления распределенными файлами и создания необходимой инфраструктуры

2. **HDFS** – распределённая файловая система, Hadoop Distributed File System – технология хранения файлов на различных серверах данных (узлах, DataNodes), адреса которых находятся на специальном сервере имен (мастере, NameNode) **.** За счет дублирования (репликации) информационных блоков, HDFS обеспечивает надежное хранение файлов больших размеров, поблочно распределённых между узлами вычислительного кластера;

3. **YARN** – система планирования заданий и управления кластером (Yet Another Resource Negotiator), которую также называют MapReduce 2.0 (MRv2) – набор системных программ (демонов), обеспечивающих совместное использование, масштабирование и надежность работы распределенных приложений**.** Фактически, YARN является интерфейсом между аппаратными ресурсами кластера и приложениями, использующих его мощности для вычислений и обработки данных

4. **Hadoop MapReduce** – платформа программирования и выполнения распределённых MapReduce-вычислений, с использованием большого количества компьютеров (узлов, nodes), образующих кластер.



Архитектурная концепция экосистемы Hadoop

Сегодня вокруг Hadoop существует целая экосистема связанных проектов и технологий, которые используются для интеллектуального анализа больших данных (Data Mining), в том числе с помощью машинного обучения (Machine Learning).

Технология хадуп появилась почти 15 лет назад и постоянно развивается. Далее показаны основные вехи ее истории:

2005 – публикация сотрудников Google Джеффри Дина и Санжая Гемавата о вычислительной концепции MapReduce сподвигла Дуга Каттинга на инициацию проекта. Разработку в режиме частичной занятости вели Дуг

Каттинг и Майк Кафарелла, чтобы построить программную инфраструктуру распределённых вычислений для свободной программной поисковой машины на Java. Свое название проект получил в честь игрушечного слонёнка ребёнка основателя **[1]**. Именно поэтому хадуп неформально называют «железный слон» и изображают его в виде этого животного.

2006 – корпорация Yahoo пригласила Каттинга возглавить специально выделенную команду разработки инфраструктуры распределённых вычислений, благодаря чему Hadoop выделился в отдельный проект.

2008 – Yahoo запустила кластерную поисковую машину на 10 тысяч процессорных ядер под управлением Hadoop, который становится проектом верхнего уровня системы проектов Apache Software Foundation. Достигнут мировой рекорд производительности в сортировке данных: за 209 секунд кластер из 910 узлов обработал 1 Тбайт информации. После этого технологию внедряют Last.fm, Facebook, The New York Times, облачные сервисы Amazon EC2

2010 – корпорация Google предоставила Apache Software Foundation права на использование технологии MapReduce. Hadoop позиционируется как ключевая технология обработки и хранения больших данных (Big Data). Начала формироваться Hadoop-экосистема: возникли продукты Avro, HBase, Hive, Pig, Zookeeper, облегчающие операции управления данными и распределенными приложениями, а также анализ информации

2011 – получение ежегодной инновационной награды медиагруппы Guardian за универсальный подход к хранению и обработке распределенных данных («швейцарский нож XXI века»)

2013 – появление модуля YARN в релизе Hadoop 2.0 значительно расширяет парадигму MapReduce, повышая надежность и масштабируемость распределенных систем

Выделяют несколько областей применения технологии

- поисковые и контекстные механизмы высоконагруженных веб-сайтов и интернет-магазинов (Yahoo!, Facebook, Google, AliExpress, Ebay и т.д.), в т.ч. для аналитики поисковых запросов и пользовательских логов;
- хранение, сортировка огромных объемов данных и разбор содержимого чрезвычайно больших файлов;
- быстрая обработка графических данных, например, газета New York Times с помощью хадуп и Web-сервиса Amazon Elastic Compute Cloud (EC2) всего за 36 часов преобразовала 4 терабайта изображений (TIFF-картинки размером в 405 КБ, SGML-статьи размером в 3.3 МБ и XML-файлы размером в 405 КБ) в PNG-формат размером по 800 КБ.

Задание:

Создать кластер Hadoop и отработать основные команды для работы с распределенной файловой системой Hadoop. Провести анализ кластера. Подключиться к кластеру с помощью различных языков и утилит.

Технология выполнения работы:

1. Установите docker и docker-compose, Docker Desktop. Создайте docker-compose файл для создания кластера. Поднимите кластер Hadoop. Проверьте работоспособность кластера в Docker Desktop. Зайдите в Namenode UI через локальный хост.
2. Откройте консоль контейнера с Hadoop с помощью docker exec и попробуйте следующие команды:

- Создать/скопировать/удалить папку;
- Добавить в HDFS любой файл;
- Скопировать/удалить этот файл;
- Просмотреть размер папки;
- Получить информацию по файловой системе (см. команду fsck);
- Установить нестандартный фактор репликации (см. команду setrep)

3. Попробовать работу с кластером различными способами:

- Используя утилиту CURL;
- Используя python3;
- Используя libhdfs3;
- Используя snakebite

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Создан кластер Hadoop;
- Отработаны основные команды;
- Предоставлен отчет о проделанной работе

Контрольные вопросы:

1. Что такое Hadoop?
2. Что такое Hadoop HDFS?
3. Перечислите основные команды работы с HDFS
4. Что такое docker и docker-compose?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 11

Тема практического занятия: Миграция данных в реляционные и нереляционные БД, HDFS-совместимые хранилища

Цель практического занятия: Научиться работать с Hadoop через расширения, поддерживающие SQL-подобные языки.

В результате выполнения данной работы обучающийся должен уметь:

- работать с Apache Hive;
- выполнять запросы с помощью языка HiveQL;
- делать выгрузку данных из Hadoop HDFS;
- создавать таблицы с различной степенью сжатия

знать:

- что такое Apache Hive;
- принципы работы и основные команды Hive;
- виды таблиц в Hive;
- операторы в HiveQL

Перечень оборудования, необходимого для выполнения задания:

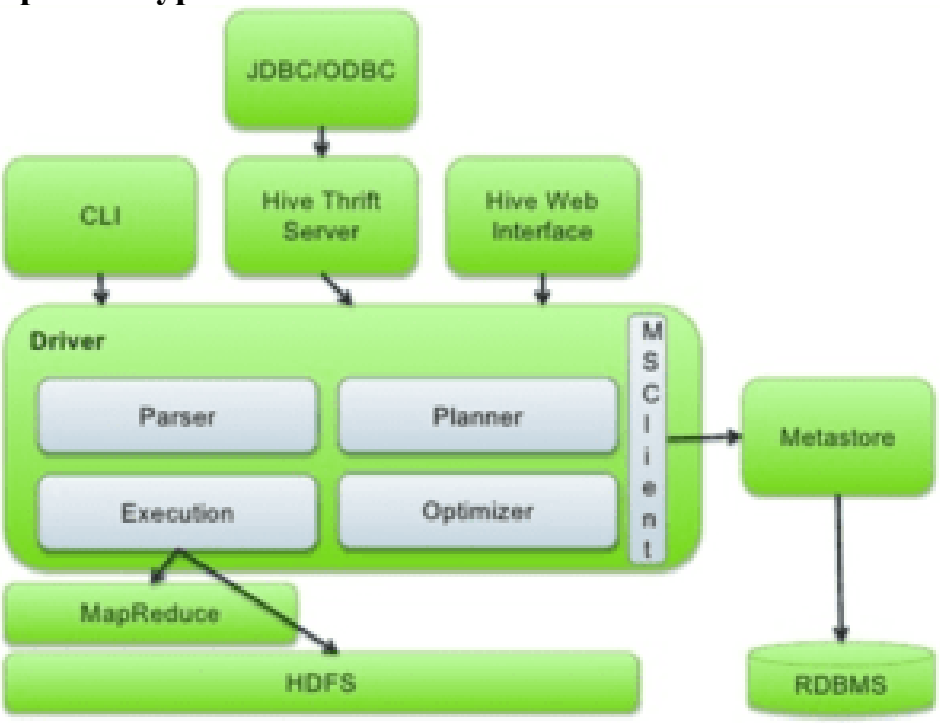
- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Apache Hive — это SQL интерфейс доступа к данным для платформы **Apache** Hadoop. **Hive** позволяет выполнять запросы, агрегировать и анализировать данные используя SQL синтаксис. Для данных в файловой системе HDFS используется схема доступа на чтение, позволяющая обращаться с данными, как с обыкновенной таблицей или реляционной СУБД. Запросы HiveQL транслируются в Java-код заданий MapReduce.

Запросы **Hive** создаются на языке запросов **HiveQL**, который основан на языке SQL, но не имеет полной поддержки стандарта SQL-92. Однако, этот язык позволяет программистам использовать их собственные запросы, когда неудобно или неэффективно использовать возможности HiveQL. HiveQL может быть расширен с помощью пользовательских скалярных функций (UDF), агрегаций (UDAF кодов), и табличных функций (UDTF).

Архитектура HIVE:



Название компонента	Описание
UI Пользовательский интерфейс	Позволяет выполнять запросы и команды в Hive: - Hive Web UI - командная строка Hive CLI или **Beeline** - Hive HD Insight (на сервере Windows) - Apache **Zeppelin** или **HUE** server
Meta Store (Хранилище мета-данных)	Хранит метаданные для таблиц **Hive** — схему на чтение (**schema-on-read**) , расположение, информацию о столбцах в таблице, типы данных, **ACL** и тд.
Hive QL Process Engine (процессор HiveQL)	Вместо написания программ **MapReduce** на **Java** мы можем написать запрос на **HiveQL** для дальнейшей компиляции и исполнения задания **MapReduce**
Execution Engine (Механизм выполнения)	Составной часть процесса **HiveQL Engine** и **MapReduce** яв ляется механизм выполнения **Hive**. Механизм выполнения обрабатывает запрос и генерирует план задач **MapReduce**.
HDFS или **HBASE**	Распределенная файловая система **Hadoop** или **HBASE** — это методы хранения данных в файловой системе

1.5 Что такое HiveQL (язык запросов Hive)?

Hive предоставляет интерфейс командной строки для написания запросов Hive с использованием языка запросов Hive (HiveQL). Обычно синтаксис HQL аналогичен синтаксису SQL , знакомому большинству аналитиков данных.

Язык Hive, основанный на SQL, отделяет пользователя от сложности программирования Map Reduce. Он использует знакомые понятия из мира реляционных баз данных, такие как таблицы, строки, столбцы и схемы, для облегчения обучения.

Большинство взаимодействий обычно происходит через интерфейс командной строки (CLI). Hive предоставляет интерфейс командной строки для написания запросов Hive с использованием языка запросов Hive (Hive-QL).

Обычно синтаксис HiveQL аналогичен синтаксису SQL, с которым знакомо большинство аналитиков данных. Hive поддерживает четыре формата файлов: TEXTFILE, SEQUENCEFILE, ORC и RCFILE (Record Columnar File).

- Для хранения метаданных для одного пользователя Hive использует базу данных derby.
- Для многопользовательских метаданных или общих метаданных Hive использует MYSQL

1.6 Встроенные операторы

Hive предоставляет встроенные операторы для операций с данными, которые должны быть реализованы в таблицах, находящихся внутри хранилища Hive.

Эти операторы используются для математических операций над операндами и возвращают определенное значение в соответствии с применяемой логикой.

Мы используем реляционные операторы для сравнения отношений между двумя операндами.

- Такие операторы, как равно, не равно, меньше, больше, чем … и т. Д.
- Все типы операндов — это числовые типы в этих операторах.

Следующая таблица даст нам подробную информацию о реляционных операторах и их использовании.

Встроенный оператор	**Описание**	**Операнд**
X = Y	TRUE, если выражение X эквивалентно выражению Y, иначе FALSE.	Он принимает все примитивные типы
X! = Y	TRUE, если выражение X не эквивалентно выражению Y, иначе FALSE.	Он принимает все примитивные типы
X <Y	TRUE, если выражение X меньше, чем Y, иначе FALSE.	Он принимает все примитивные типы
X <= Y	TRUE, если выражение X меньше или равно выражению Y, иначе FALSE.	Он принимает все примитивные типы
X> Y	TRUE, если выражение X больше, чем выражение Y, иначе FALSE.	Он принимает все примитивные типы
X> = Y	TRUE, если выражение X больше или равно выражению Y, иначе FALSE.	Он принимает все примитивные типы
X НУЛЬ	TRUE, если выражение X имеет значение NULL, иначе FALSE.	Требуется все типы
X НЕ НУЛЬ	FALSE, если выражение X имеет значение NULL, иначе TRUE.	Требуется все типы
X как Y	TRUE, если строковый шаблон X соответствует Y, иначе FALSE.	Принимает только строки
X RLIKE Y	NULL, если X или Y NULL, TRUE, если любая подстрока X соответствует регулярному выражению Java Y, иначе FALSE.	Принимает только строки
X REGEXP Y	Так же, как RLIKE.	Принимает только строки

Арифметические операторы:

Мы используем арифметические операторы для выполнения арифметических операций над операндами

- Арифметические операции, такие как сложение, вычитание, умножение и деление между операндами, мы используем эти операторы.
- Все типы операндов являются типами чисел в этих Операторах.

Пример примера:

2 + 3 дает результат 5.

В этом примере «+» — оператор, а 2 и 3 — операнды. Возвращаемое значение 5

Следующая таблица даст нам подробную информацию об арифметических операторах

Встроенный оператор	Описание	Операнд
X + Y	Он вернет результат сложения значений X и Y.	Требуется все типы чисел
X — Y	Он вернет результат вычитания Y из значения X.	Требуется все типы чисел
X * Y	Он вернет результат умножения значений X и Y.	Требуется все типы чисел
X / Y	Он вернет результат деления Y на X.	Требуется все типы чисел
X% Y	Он вернет остаток, полученный в результате деления X на Y.	Требуется все типы чисел
X & Y	Он вернет выход побитового И X и Y.	Требуется все типы чисел
X | Y	Он вернет вывод побитового ИЛИ X и Y.	Требуется все типы чисел
X ^ Y	Он вернет вывод побитового XOR X и Y.	Требуется все типы чисел
~ X	Он вернет выход побитового НЕ X.	Требуется все типы чисел

Логические операторы:

Мы используем логические операторы для выполнения логических операций над операндами

- Логические операции, такие как И, ИЛИ, НЕ между операндами, мы используем эти операторы.
- Все типы операндов имеют тип BOOLEAN в этих операторах.

Следующая таблица даст нам подробную информацию о логических операторах

Операторы	Описание	Операнды
X И Y	TRUE, если X и Y TRUE, иначе FALSE.	Только логические типы
X && Y	То же, что X и Y, но здесь мы используем символ &&	Только логические типы
X ИЛИ Y	TRUE, если X или Y или оба TRUE, иначе FALSE.	Только логические типы

X || Y	**То же, что X ИЛИ Y, но здесь мы используем || символ**	**Только логические типы**
НЕ X	**ИСТИНА, если X ЛОЖЬ, иначе ЛОЖЬ.**	**Только логические типы**
!ИКС	**То же, что НЕ X, но здесь мы используем! символ**	**Только логические типы**

Операторы на сложных типах:

Следующая таблица даст нам подробную информацию об операторах сложного типа. Это операторы, которые предоставляют другой механизм для доступа к элементам сложных типов.

Операторы	Операнды	Описание
A [n]	**A является массивом, а n является целочисленным типом**	**Он вернет n-й элемент в массиве A. Первый элемент имеет индекс 0**
M [ключ]	**M — это карта <K, V>, а ключ имеет тип K**	**Возвращает значения принадлежащие ключу на карте**

Конструкторы сложного типа:

Следующая таблица даст нам подробную информацию о конструкторах сложного типа. Он будет создавать экземпляры на сложных типах данных. Это сложные типы данных, такие как Array, Map и Struct в Hive.

В этом разделе мы рассмотрим операции, выполняемые над конструкторами сложного типа.

Операторы	Операнды	Описание
массив	**(val1, val2, …)**	**Это создаст массив с заданными элементами, как упомянуто как val1, val2**
Create_union	**(тег, val1, val2, …)**	**Это создаст тип объединения со значениями, на которые указывает параметр tag**
карта	**(ключ1, значение1, ключ2, значение2, …)**	**Это создаст карту с заданными парами ключ / значение, упомянутыми в операндах**
Named_struct	**(name1, val1, name2, val2, …)**	**Это создаст Struct с заданными именами полей и значениями, упомянутыми в операндах**
STRUCT	**(val1, val2, val3, …)**	**Создает структуру с заданными значениями поля. Имена полей структуры будут col1, col2,.**

Задание:

Подключить Hive к кластеру Hadoop и отработать основные команды работы с Hive. Создать таблицы разного формата с компрессией и без нее. Выполнить запросы к таблицам. Сравнить результаты запросов. Подготовить итоговый отчет.

Технология выполнения работы:

1. Подключите Hive через docker, проверьте работоспособность.
2. Выполните задания:

a. Создать таблицы в форматах PARQUET/ORC/AVRO с компрессией и без оной. (Выберите один вариант, например ORC с компрессией);
b. Заполнить таблицы данными из большой БД;
c. Получить информацию по раз меру данных;
d. Посчитать count некоторых колонок в разных форматах хранения;
e. - Посчитать агрегаты по одной и нескольким колонкам в разных форматах.

3. Сделать выводы о эффективности хранения и компресии.
4. Составьте отчет о проделанной работе.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Добавлено расширение Hive в кластер Hadoop
- Выполнены задания по работе с данными
- Предоставлен итоговый отчет

Контрольные вопросы:

1. Что такое Hive?
2. Какие виды таблиц есть в Hive?
3. Как узнать размер таблицы в Hive?
4. Перечислите основные команды для работы с Hive

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 12

Тема практического занятия: Загрузка потоковых данных в Hadoop

Цель практического занятия: ознакомление студентов с методами и инструментами, используемыми для эффективной обработки потоковых данных в экосистеме Hadoop

В результате выполнения данной работы обучающийся должен уметь:

- загружать данные в Hadoop HDFS;
- создавать пайплайн обработки потоковых данных;
- работать с Hadoop с поиощью языка Python;

знать:

- способы загрузки данных в Hadoop
- способы потоковой обработки данных

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Потоковая обработка данных - это метод обработки данных в режиме реального времени, где данные обрабатываются по мере их поступления.

Информация записывается в потоковом режиме, за счет чего достигается высокая пропускная способность. Клиент, осуществляющий запись, кэширует данные во временном локальном файле, пока их объем не достигнет размера одного HDFS-блока (по умолчанию 64 MB).

Накопив данные на один блок, клиент отправляет на сервер имен NameNode запрос на создание файла, указав размер блока для создаваемого файла и количество реплик.

Сервер имен регистрирует новый файл, выделяет блок и возвращает клиенту идентификаторы узлов данных для хранения копий (реплик) этого HDFS-блока. Также сервер имен уведомляет другие узлы данных, на которые будут писаться реплики файлового блока.

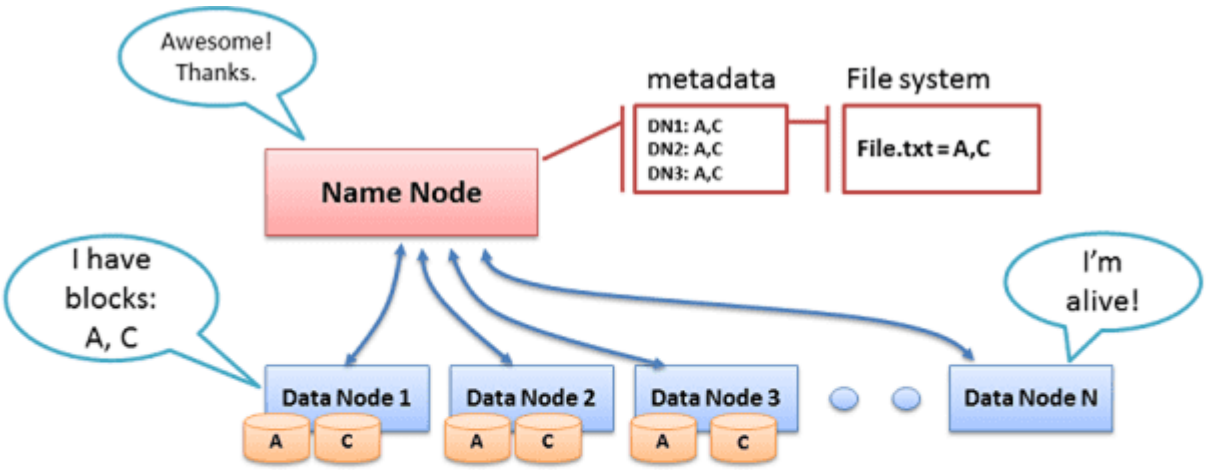
Все блоки в HDFS, кроме последнего блока файла, имеют одинаковый размер. Каждый блок может быть размещён на нескольких узлах. Размер блока и коэффициент репликации – число узлов для размещения каждого HDFS-блока – определяются в файловых настройках. Репликация обеспечивает устойчивость системы к отказам отдельных узлов

Клиент начинает передачу данных блока из временного файла первому по списку узлу данных, который сохраняет информацию на своем диске и пересылает ее следующему серверу данных. Второй узел передает данные третьему и далее: файловый поток передается в конвейерном режиме и автоматически реплицируется на нужном количестве узлов. Все HDFS-блоки реплицируются столько раз, сколько было указано клиентом (по умолчанию 3). Для повышения надежности узлы для хранения 2-ой и 3-ей реплики располагаются в разных серверных стойках. Расположение следующих реплик вычисляется произвольно. Для защиты от сбоев можно настроить кластер так, чтобы сервер имен знал, на каких серверных стойках расположены узлы данных. Для этого используется специальный механизм Hadoop — **rack awareness**.

По завершении записи HDFS-блока каждый узел данных из цепочки в обратном порядке (т.е. с конца) отправляет клиенту сообщения об успешной операции. После этих подтверждений клиент оповещает сервер имен об успешной записи блока и получает цепочку узлов данных для записи 2-го блока и т.д.

Если сервер имен не принимает от узла данных heartbeat-сообщений, он считает этот DataNode вышедшим из строя (умершим) и реплицирует данные, хранящиеся на этом узле, на другие сервера данных (живые) **[3]**. Если клиент смог успешно записать блок хотя бы на 1 узел, можно не беспокоиться за дальнейшую репликацию – это находится в сфере ответственности сервера имен, который сам обеспечит распространение информации на нужном уровне

Окончив запись, клиент уведомляет сервер имен NameNode, который фиксирует транзакцию создания файла в своем журнале. После этого HDFS-файл становится доступным для использования: чтения, повторной репликации или удаления



- Data Node sends Heartbeats
- Every 10th heartbeat is a Block report
- Name Node builds metadata from Block reports
- TCP – every 3 seconds
- If Name Node is down, HDFS is down

Запись блоков большого файла Big Data в HDFS

Задание:

Создать конвейер обработки потоковых данных и их сохранение в распределенной файловой систем Apache Hadoop. Провести оценку собранных данных. Провести анализ архитектуры созданного конвейера данных. Подготовить итоговый отчет.

Технология выполнения работы:

1. Разверните кластер Hadoop
2. В Интернете найдите любой источник данных, подходящий для парсинга или открытое Api.
3. Обеспечьте потоковый сбор данных
4. Обеспечьте потоковую валидацию и обработку данных
5. Обеспечьте поступление потоковых данных в Hadoop HDFS
6. Оцените объем и качество собранных данных, напишите с какими трудностями вы столкнулись.
7. Составьте отчет о проделанной работе. Прикрепите исходники кода, а также выгрузку из Hadoop HDFS в формате csv.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Создан конвейер обработки потоковых данных;
- Сделан отчет о проделанной работе;
- Сделана выгрузка данных из HDFS в csv

Контрольные вопросы:

1. Что такое потоковая обработка данных?
2. Какие преимущества предоставляет потоковая обработка данных по сравнению с пакетной?
3. Что такое Hadoop HDFS?
4. Каким образом происходит сохранение данных в Hadoop HDFS?
5. Какие основные шаги включает в себя процесс потоковой обработки данных в Hadoop?
6. Что такое Apache Kafka и как он связан с потоковой обработкой данных?
7. Какие инструменты или библиотеки используются для реализации потоковой обработки данных в Hadoop экосистеме?
8. Какие виды задач можно эффективно решать с использованием потоковой обработки данных в Hadoop?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 13

Тема практического занятия: Анализ данных сервисами реляционных и нереляционных хранилищ

Цель практического занятия: ознакомление с основными принципами работы реляционных и нереляционных баз данных, изучение их особенностей, а также практическое применение инструментов для анализа данных в этих системах.

В результате выполнения данной работы обучающийся должен уметь:

- анализировать данные средствами реляционных СУБД;
- анализировать данные средствами нереляционных СУБД;
- строить модели машинного обучения средствами СУБД;
- работать с ClickHouse

знать:

- базовые приемы анализа данных средствами систем управления базами данных;
- команды для работы с ClickHouse

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

ClickHouse – колоночная реляционная СУБД с открытым исходным кодом от компании Яндекс для быстрой обработки аналитических SQL-запросов на структурированных больших данных (Big Data) в режиме реального времени.

1.7 ИСТОРИЯ РАЗРАБОТКИ И РАЗВИТИЯ

Основными ключевыми вехами в истории ClickHouse считаются следующие:

- **2009 год** – компания Яндекс разработала первый прототип своей аналитической СУБД для собственных нужд, в рамках проекта веб-аналитики «Яндекс.Метрика» с целью построения отчетов в режиме реального времени по неагрегированным логам пользовательских действий
- **2013 год –** использование СУБД для анализа метаданных о событиях эксперимента на одном из детекторов Большого андронного коллайдера в CERN
- **2014 год** – Яндекс полностью перезапустил свой сервис веб-аналитики Метрика 2.0 на базе ClickHouse, благодаря чему пользователи могут строить произвольные отчеты
- **2016 год –** переход ClickHouse из проприетарного решения в open source – Яндекс опубликовал исходный код СУБД под лицензией Apache 2.0
- **2019 год** – ClickHouse включен в состав реестра отечественного программного обеспечения, что позволяет использовать эту СУБД в проектах цифровизации государственных и частных компаний РФ с учетом требований к импортозамещению
- **2019 год** – компания «Аренадата Софтвер», разработчик первого отечественного дистрибутива Apache Hadoop и других корпоративных решений для хранения и обработки Big Data, выпустил на базе ClickHouse собственную аналитическую СУБД Arenadata QuickMarts. Продукт адаптирован для нужд сектора enterprise и включает возможности гибкой авторизации пользователей с разграничением доступа, поддержку формата ORC и интеграцию с безопасным протоколом Kerberos для экосистемы Hadoop.

1.8 АРХИТЕКТУРА И ПРИНЦИПЫ РАБОТЫ CLICKHOUSE

Ключевым преимуществом Кликхаус считается **высокая скорость выполнения SQL-запросов** на чтение (OLAP-сценарий), которая обеспечивается благодаря следующим архитектурным особенностям:

- **столбцовое хранение данных**, что позволяет считывать данные только из нужных колонок и эффективно сжимать однотипную информацию;
- **физическая сортировка данных по первичному ключу** позволяет быстро получать конкретные значения или диапазонов;
- **векторные вычисления** по кусочкам столбцов снижают издержки на диспетчеризацию и позволяют более эффективно использовать CPU;
- **распараллеливание операций** как в пределах одного сервера на несколько процессорных ядер, так и в рамках распределенных вычислений на кластере за счет механизма шардирования;
- **поддержка приближенных вычислений** на части выборки, что снижает число обращений к жесткому диску и еще больше повышает скорость обработки данных.

Стоит отметить, что в отличие от других популярных столбцовых СУБД для Big Data, например, SAP HANA и Google PowerDrill, которые работают только в оперативной памяти, ClickHouse работает с жесткими дисками. Это снижает **стоимость эксплуатации системы**, поскольку жесткие диски дешевле RAM.

При работе в кластере данные реплицируются асинхронно в фоновом режиме с поддержкой полной идентичности на разных репликах. Apache ZooKeeper используется для координации процесса репликации, но не участвует в обработке данных и выполнения запросов. При сбое в большинстве случаев восстановление данных происходит автоматически. По желанию можно включить кворумную запись данных. Кластер Кликхаус масштабируется линейно путем добавления новых узлов. ClickHouse поддерживает диалект SQL с расширениями, такими как массивы и вложенные структуры данных, вероятностные структуры, возможность подключить внешнее key-value хранилище. Еще СУБД содержит множество возможностей интеграции с другими Big Data системами, такими как Apache Kafka и HDFS, а также MySQL и прочие внешние источники данных через ODBC или JDBC

При том, что Кликхаус является реляционной СУБД, он не поддерживает транзакции, а также точечные операции UPDATE и DELETE. Кроме того, в данной системе отсутствуют оконные функции и полноценный оптимизатор запросов. Подробнее о достоинствах и недостатках ClickHouse мы рассказываем в отдельных статьях.

1.9 ГДЕ ИСПОЛЬЗУЕТСЯ КЛИКХАУС: КОМПАНИИ И BIG DATA ПРОЕКТЫ

Благодаря высокой скорости генерации аналитических отчетов по большим данным в режиме реального времени, ClickHouse наиболее востребован в следующих областях:

- веб-аналитика и контекстная реклама;
- real time мониторинг бизнес-метрик, например, анализ потребительского поведения на сайте;
- интерактивное взаимодействие с пользователями, к примеру, онлайн-игры;
- контроль технических показателей, в т.ч. интернет вещей (Internet of Things);
- реализация корпоративных хранилищ данных, например, как это сделано в Ситимобил.

При том, что ClickHouse изначально был разработан в Яндексе, он применяется не только в сервисах этого ИТ-гиганта. Сегодня множество отечественных и зарубежных компаний используют эту Big Data СУБД для быстрой аналитики больших данных. Из наиболее известных внедрений стоит упомянуть иностранные Cloudflare и Bloomberg, отечественную соцсеть ВКонтакте, Тинькофф банк, сервис онлайн-объявлений Avito, новостную сеть СМИ2, онлайн-кинотеатр ivi.ru, интернет-порталы Mail.ru и Rambler

Задание:

Создать колоночную базу данных в ClickHouse и провести анализ данных в СУБД. Выполнить запросы на аггрегацию статистических данных. Построить модели машинного обучения с помощью ClickHouse. Выполнить оценку проведенной работы и сделать выводы. Подготовить итоговый отчет о проделанной работе.

Технология выполнения работы:

1. Создайте базу данных в ClickHouse. Создайте таблицу с данными о потоке пассажиров на станциях (любой вид перевозки людей).
2. Выполните аггрегацию данных:
   - Найдите среднее количество пассажиров по каждому часу
   - Найдите моду количества пассажиров по каждому часу
   - Найдите самый загруженный час на каждой станции
   - Найдите наименее нагруженный час на каждой станции
3. Постройте различные модели регрессии средствами ClickHouse для предсказания пассажиропотока. Сравните модели и запишите результаты в таблицу.
4. Создайте отчет о проделанной работе. В отчете напишите отзыв о ClickHouse. Сравните анализ данных средствами СУБД и средствами машинного обучения.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Создана база данных;
- Выполнены запросы на аггрегацию данных;
- Построены модели регрессии;
- Создан итоговый отчет

Контрольные вопросы:

1. Каким образом проводится анализ данных в NoSQL базах данных?
2. Каким образом проводится анализ данных в SQL базах данных?
3. Что такое ClickHouse?
4. Как проводить аггрегацию данных средствами СУБД?
5. Как строить модели средствами СУБД?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 14

Тема практического занятия: Работа с Kafka

Цель практического занятия: Научиться работать с Apache Kafka

В результате выполнения данной работы обучающийся должен уметь:

- работать с Apache Kafka;
- развертывать Apache Kafka;
- разрабатывать приложения с применением Apache Kafka

знать:

- основные понятия Apache Kafka;
- основные принципы работы Kafka;
- основные концепции Apache Kafka

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Kafka Apache — распределенная система обмена сообщениями между серверными приложениями в режиме реального времени. Благодаря высокой пропускной способности, масштабируемости и надежности применяется в компаниях, работающих с большими объемами данных. Написана на языках Java и Scala.

Kafka разработана компанией LinkedIn. В 2011 году разработчик опубликовал исходный код системы. С тех пор платформа развивается и поддерживается как открытый проект в рамках фонда Apache Software Foundation. Apache Kafka используют многие крупные компании, такие как LinkedIn, Microsoft, The New York Times, Netflix и другие.

Kafka Apache — эффективный инструмент для организации работы серверных проектов любого уровня. Благодаря гибкости, масштабируемости и отказоустойчивости используется в различных направлениях IT-индустрии, от сервисов потоковых видео до аналитики Big Data.

Для связи микросервисов. Kafka — связующее звено между отдельными функциональными модулями большой системы. Например, с ее помощью

можно подписать микросервис на другие компоненты для регулярного получения обновлений.

Потоковая передача данных. Высокая пропускная способность системы позволяет поддерживать непрерывные потоки информации. За счет грамотной маршрутизации «Кафка» не только надежно передает данные, но и позволяет производить с ними различные операции.

Ведение журнала событий. Kafka сохраняет данные в строго организованную структуру, в которой всегда можно отследить, когда произошло то или иное событие. Информация хранится в течение заданного промежутка времени, что можно использовать для разгрузки базы данных или медленно работающих систем логирования.

Кратко архитектуру системы сообщений можно охарактеризовать следующим образом:

- распределенность — отдельные узлы системы располагаются на нескольких аппаратных платформах (кластерах). Это обеспечивает ей высокую отказоустойчивость;
- масштабируемость — систему можно наращивать за счет простого добавления новых узлов (брокеров сообщений).

В архитектуре Kafka Apache ключевыми являются концепции:

- продюсер (producer) — приложение или процесс, генерирующий и посылающий данные (публикующий сообщение);
- потребитель (consumer) — приложение или процесс, который принимает сгенерированное продюсером сообщение;
- сообщение — пакет данных, необходимый для совершения какой-либо операции (например, авторизации, оформления покупки или подписки);
- брокер — узел (диспетчер) передачи сообщения от процесса-продюсера приложению-потребителю;
- топик (тема) — виртуальное хранилище сообщений (журнал записей) одинакового или похожего содержания, из которого приложение-потребитель извлекает необходимую ему информацию.

В упрощенном виде работа Kafka Apache выглядит следующим образом:

Приложение-продюсер создает сообщение и отправляет его на узел Kafka.

Брокер сохраняет сообщение в топике, на который подписаны приложения-потребители.

Потребитель при необходимости делает запрос в топики и получает из него нужные данные.

Сообщения хранятся в Kafka в виде журнала коммитов — записей, размещенных в строгой последовательности. Их можно только добавлять. Удалять или корректировать нельзя. Сообщения хранятся в той последовательности, в которой поступили, их считывание ведется слева направо, а отслеживание — по изменению порядкового номера. Брокеры Kafka не обрабатывают записи — только помещают их в тему на кластере.

Хранение может длиться в течение определенного периода или до достижения заданного порога.

Если тема слишком разрастается, для упрощения и ускорения процесса она разделяется на секции. Каждая секция содержит сообщения, сгруппированные по объединяющему признаку. Например, массив пользовательских запросов можно сгруппировать по первой букве имени пользователей. Так приложению-потребителю не придется просматривать весь топик — только нужную тему, что ускоряет процесс обмена сообщениями.

Kafka — распределенная система обмена сообщениями, узлы которой содержатся на нескольких кластерах. Принимая сообщение от продюсера, она реплицирует (копирует) его, а копии сохраняет на разных узлах. При этом один из брокеров назначается ведомым в секции, через него потребители будут обращаться к записям. Другие брокеры остаются ведомыми, их главная задача — обеспечить сохранность сообщения (его копий) даже при выходе одного или нескольких узлов из строя. Распределенный характер и механизм репликации записей обеспечивают системе высокую устойчивость. Надежность повышает интеграция с Apache ZooKeeper, которая обеспечивает координацию компонентов друг с другом.

Apache Kafka поддерживает «горячее» расширение, то есть ее можно увеличивать с помощью простого добавления новых машин в кластеры, не отключая всю систему. Так исключаются простои, связанные с переоборудованием серверных мощностей. Принцип удобнее горизонтального масштабирования, при котором на одну серверную машину «навешиваются» дополнительные ресурсы: жесткие диски, CPU, RAM и т.д. При необходимости систему можно легко сократить, исключив лишние машины из кластера.

В Kafka процессы генерирования/отправки и считывания сообщений организованы независимо друг от друга. Тысячи приложений, процессов могут одновременно и параллельно играть роль генераторов и потребителей сообщений. В сочетании с распределенным характером и масштабируемостью это позволяет применять «Кафка» как в небольших, так и в масштабных проектах с большими объемами данных.

Kafka распространяется по свободной лицензии фонда Apache Software Foundation. Благодаря этому Kafka Apache имеет ряд преимуществ:

- большой объем подробной справочной информации от официальных разработчиков, а также различных мануалов, лайфхаков, инструкций и обзоров от большого числа энтузиастов-любителей и профессионалов;
- большое количество дополнительных программных пакетов, патчей от сторонних разработчиков, расширяющих и улучшающих базовый функционал системы;
- возможность самостоятельно адаптировать систему под специфику проекта за счет гибкости настроек.

В Kafka есть инструменты, обеспечивающие безопасную работу и достоверность данных. Например, настроив уровень изоляции для транзакций, можно исключить чтение незавершенных или отмененных сообщений. Кроме того, благодаря сохранению данных в топиках пользователь может в любой

момент отследить изменения в системе. А принцип последовательной записи позволяет быстро находить нужные сообщения.

Данные в Kafka сохраняются в долговременные виртуальные хранилища в течение заданного периода времени (дней, недель, месяцев). За счет распределенного хранения информации она не потеряется при сбое одного или нескольких узлов, и потребитель сможет в любой момент обратиться к нужному сообщению в топике, отследив его смещение.

Благодаря собственному протоколу на базе TCP «Кафка Апач» взаимодействует с другими протоколами передачи данных (REST, HTTP, XMPP, STOMP, AMQP, MQTT). Встроенный фреймворк Kafka Connect позволяет Kafka подключаться к базам данных, файловым и облачным хранилищам.

Единственный заметный недостаток системы — ориентированность на обработку больших объемов данных. Из-за этого функционал маршрутизации потоков ограничен по сравнению с другими аналогичными платформами. По мере развития Kafka это различие становится менее заметным, а сама система — более гибкой и универсальной.

Задание:

Развернуть Apache Kafka и подготовить его для локальной разработки. Написать два микросервиса согласно техническому заданию, указанному в этапах выполнения практического задания. Реализовать общение микросервисов с помощью Apache Kafka. Провести докеризацию микросервисов. Подготовить итоговый отчет о проделанной работе.

Технология выполнения работы:

1. Выполните установку Apache Kafka
2. На любом веб-фреймворке напишите два микросервиса. Первый микросервис отвечает за регистрацию пользователя в системе. Второй микросервис отвечает за отправку уведомлений пользователю. Реализуете регистрацию пользователя с помощью email. Необходимо проводить верификацию email с помощью отправки кода подтверждения. Также отправьте приветственное письмо после регистрации. Отправка сообщений должна быть реализована на втором микросервисе. Общение микросервисов происходит через Kafka.
3. Оберните Kafka и микросервисы в docker-compose
4. Проведите сравнение Kafka с брокером сообщений RabbitMQ
5. Создайте отчёт о проделанной работе.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Реализованы микросервисы и паттерн consumer-subscriber с помощью Kafka;
- Проведено сравнение Kafka с брокерами сообщений;
- Создан итоговый отчет

Контрольные вопросы:

1. Что такое Apache Kafka и для чего он используется?
2. Какие компоненты включает в себя архитектура Kafka?
3. Что такое Topic в Kafka?
4. Какие роли выполняют Producer и Consumer в системе Kafka?
5. Какие гарантии по доставке сообщений предоставляет Kafka?
6. Что такое Partition в Kafka?
7. Какую роль играет Zookeeper в архитектуре Kafka?
8. Что такое Consumer Group в Kafka и для чего он используется?
9. Как можно обеспечить отказоустойчивость в Kafka?
10. Что такое Kafka Stream?
11. Какие протоколы передачи данных могут использоваться в Kafka?
12. Какие уровни надежности предоставляет Kafka для хранения сообщений?
13. Как можно управлять конфигурацией Kafka?
14. Как обеспечивается масштабируемость Kafka?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 15

Тема практического занятия: Базовые операции в Spark Structured Streaming

Цель практического занятия: Научиться работать с Apache Spark

В результате выполнения д
анной работы обучающийся должен
уметь:

- создавать потоковые DataFrame;
- создавать SQL-подобные запросы для работы с потоковыми данными;
- применять агрегации и оконные функции в Spark Structured Streaming;
- обеспечивать отказоустойчивость с использованием проверки точек сохранения

знать:

- архитектуру Apache Spark;
- способы создания потоковые DataFrame;
- особенности применения библиотеки PySpark;
- оконные операции в контексте потоковой обработки данных

` Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Библиотека потоковой обработки событий Structured Streaming основана на механизме Spark SQL, который обрабатывает потоковые данные постепенно и непрерывно по мере их поступления. Поскольку данные могут приходить из источника с опозданием, возникает задержка между исходным временем происхождения события в реальном мире и его поступлением в обработку. Для этого в Structured Streaming есть механизм водяных знаков (watermark), который позволяет поддерживать состояние поступающих данных, сохранять их в памяти и точно обновлять, объединяя их с данными, поступившими с опозданием. Чтобы выполнять этот запрос в течение нескольких дней, системе должна быть известна информация о накоплении состояния в памяти. Это связано с тем, что приложение должно знать, когда оно перестанет получать

просроченные данные, чтобы упростить сбор данных. Проще говоря, поздние данные в пределах порога будут агрегированы, но данные после порога начнут отбрасываться. Внутри Spark Structured Streaming это обрабатывается следующим образом: водяной знак для каждого микропакета вычисляется в конце предыдущего пакета. Это означает, что водяной знак рассчитывается для каждого микропакета еще до начала выполнения. В случае нескольких потоков Spark отслеживает самый высокий водяной знак среди них. Сначала рассчитываются индивидуальные водяные знаки, а минимальное значение выбирается позже в качестве глобального водяного знака, который используется для исключения событий.

Существуют некоторые ограничения на использование водяных знаков, которые должны быть соблюдены:

- Spark поддерживает несколько режимов вывода, о которых мы писали здесь. Из них только 2, *append (добавить)* и *update (обновить)* поддерживаются для водяных знаков;
- функцию *withWatermark()* можно вызывать только для того же столбца, который используется в агрегате. Этот метод надо вызывать перед агрегацией, чтобы использовать детали водяного знака. Поэтому, например, *withWatermark(«time», «1 min»).groupBy(«time2»).count()* не будет действовать в режиме вывода с добавлением, т.к. водяной знак не определен в столбце агрегации. Аналогично, *df.groupBy(«time»).count().withWatermark(«time», «1 min»)* тоже не будет действовать в режиме вывода *append*.

Помимо этих ограничений, также разработчики Spark-приложения стоит помнить о других особенностях Structured Streaming, связанных с версиями этого вычислительного движка. В частности, Apache Spark 2.3.2 не поддерживает стандартный сбор и визуализацию метрик. Поэтому придется прикреплять к приложению класс слушателя, который извлекает статистику из каждого микропакета и сохраняет ее, например, в формате JSON в HDFS. Про слушатели в Apache Spark мы писали в этой статье. Такие слушатели позволяют следить за работоспособностью потокового приложения, например, сколько времени заняло выполнение какого-то триггера, т. е. получение смещений, обработка данных и фиксация WAL, сколько времени заняло получение смещений для новых данных для обработки каждого из определенных источников и фиксации новых доступных смещений.

В Apache Spark 3.0 использовать такие слушатели нет необходимости, т.к. все эти системные метрики отображаются в пользовательском веб-GUI на вкладке «Потоковая передача» в режиме реального времени. Но, помимо отсутствия этой визуализации, в более ранних версиях Apache Spark есть еще пара проблем с потоковой передачей. Например, отсутствие поддержки полных внешних соединений (full outer join) и проблема согласованности данных в left join.

Кроме того, в Apache Spark 2.3 можно столкнуться с проблемой замедления больших заданий, которая обусловлена сжатием метаданных. Как ее решить, рассмотрим далее.

Контрольные точки — это способ достижения семантики строго однократной доставки сообщений (exactly once) в Apache Spark. Потоковое задание сохраняет информацию о входных и выходных пакетах в контрольной точке и фиксирует файлы для каждого микропакета. Поскольку потоковые задания выполняются круглосуточно и непрерывно, это приводит к обработке большого объема данных. Размер файлов со временем увеличивается и может достигать 10 ГБ. В сжатых файлах хранятся сведения обо всех файлах, обработанных и созданных заданием потоковой передачи с момента его начала.

Сжатие файла фиксации (_Spark_Metadata) выполняется в драйвере для каждых 10 пакетов (начиная с 0). Это единый монолитный процесс, который занимает до 10–15 минут в зависимости от размера файла сжатия. Он резко снижает производительность задания Apache Spark Structured Streaming, поскольку, пока драйвер работает над сжатием файла, все рабочие узлы ничего не делают, и задание тратит ресурсы впустую, оставаясь полностью бездействующим. Определить интервал уплотнения можно через конфигурацию *spark.sql.streaming.fileSink.log.compactInterval*.

По умолчанию это значение равно 10, т.е. после каждых 10 микропакетов происходит сжатие файла фиксации. Он добавляет дельты последних 9 пакетов к предыдущему файлу *.compact* и создает новый файл. Эти компактные файлы имеют формат JSON, а также отслеживают метки времени для файлов с другой информацией. Если объем компактного файла слишком большой и изменение интервала уплотнения не помогает, дата-инженер может написать код для очистки метаданных из каталога _Spark_Metadata. Для этого можно использовать следующий подход:

- удалить задание Spark, чтобы прочитать последний JSON-файл *.comapct* для _Spark_Metadata;
- отфильтровать все записи для старых файлов по сроку хранения;
- сохранить старый компактный файл для резервного копирования;
- переместить новый измельченный компактный файл в каталог _Spark_Metadata, с сохранением его структуры и имени;
- проверить, что задание потоковой передачи Spark не запущено, пока шло обновление компактного файла метаданных.

Задание:

Постройте модель машинного обучения и обеспечьте её дообучение на потоковых данных с помощью Spark. Обеспечьте отладку и поддержку написанной системы. Сделайте вывожы о работе со Spark. Подготовьте итоговый отчет, отражающий результаты проделанной работы.

Технология выполнения работы:

1. Напишите модель машинного обучения для классификации отзывов на нейтральные, хорошие и плохие.
2. Создайте БД с таблицей, в которую будут поступать отзывы.
3. Напишите парсинг отзывов с любого источника
4. Обеспечьте дообучение модели при поступлении новых данных в БД

5. Создайте отчёт о проделанной работе.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Реализован механизм дообучения модели с помощью Spark;
- Создан итоговый отчет

Контрольные вопросы:

1. Что такое Spark Structured Streaming?
2. Каким образом Spark Structured Streaming обеспечивает единый API для обработки данных как в потоковом, так и в пакетном режимах?
3. Как создать потоковый DataFrame в Spark Structured Streaming?
4. Какие источники данных поддерживаются в Spark Structured Streaming?
5. Как определить структуру данных (схему) в Spark Structured Streaming?
6. Каким образом определить оконные операции в Spark Structured Streaming?
7. Каким образом задать выходной источник данных (sink) в Spark Structured Streaming?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 16

Тема практического занятия: Анализ данных в Hadoop

Цель практического занятия: Научиться проводить анализ данных в Hadoop

В результате выполнения данной работы обучающийся должен уметь:

- исать запросы на Hive Query Language (HQL) для анализа данных;
- использовать Sqoop для импорта и экспорта данных между Hadoop и реляционными базами данных;
- программировать на MapReduce для обработки и анализа данных в Hadoop;
- обрабатывать и проводить агрегацию данных с использованием различных инструментов в экосистеме Hadoop

знать:

- основные инструменты Hadoop, таких как Hadoop MapReduce, Hadoop Distributed File System (HDFS), Hive, Pig, HBase и Sqoop.;
- архитектуру Hadoop, включая HDFS (Hadoop Distributed File System) и MapReduce;
- методы управления данными в Hadoop, включая загрузку, выгрузку, хранение и удаление данных;
- основы администрирования кластера Hadoop, включая мониторинг, настройку и обеспечение безопасности

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

1. **Хранение данных**

Данные хранятся в распределенной файловой системе Hadoop HDFS (Hadoop distributed file system). Благодаря этому происходит быстрое чтение данных, что особенно удобно при анализе данных. Запись данных соответственно выполняется дольше. Поэтому имеет смысл пакетной загрузки данных в максимально незагруженное время.

2. **Чтение данных**

Чтение и выгрузку данных удобнее всего производить с помощью SQL-подобных языков, которые позволяют забирать данные из HDFS. В прошлых лабораторных работах для этой цели мы использовали Apache Hive.

3. **Анализ данных**

Полученные из HDFS данные необходимо предобработать – очистить пропуски, удалить выбросы. Для этого можно воспользоваться средствами языка Python или R.

После и во время предобработки необходимо визуализировать данные для наглядного восприятия закономерностей и поиска важных фичей. Визуализировать данные можно на языке Python с помощью библиотек – matplotlib, seaborn, plotly и им подобным.

Далее, после разведочного анализа данных и предобработки появляется опциональная стадия feature engineering. На ней необходимо отобрать или дополнить признаки датасета, чтобы получить лучшее качество данных при построении ML-моделей.

После подготовки данных можно проводить обучение ML-модели.

Задание

Сделать выгрузку данных, находящихся в Apache Hadoop HDFS. Провести предобработку данных. Визуализировать данные и сделать выводы. Подготовить итоговый отчет с результатами проделанной работы. Также отразите в отчете особенности данных, полученные в результате анализа.

Технология выполнения работы:

1. Загрузите любые данные в кластер Hadoop или возьмите имеющиеся.
2. Сделайте выгрузку данных с помощью Hive
3. Проведите предобработку полученного набора данных
4. Проведите разведочный анализ данных и визуализацию
5. Выполните конструирование признаков (feature engineering), если оно необходимо
6. В отчете отразите результаты работы в п.2 – п.5. Дайте комментарии к работе и к коду. Сделайте выводы по вашим данным.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету::

- Проведен полный цикл выгрузки и подготовки данных;
- Проведен визуальный анализ данных;
- Дана интерпритация полученным результатам;
- Создан итоговый отчет

Контрольные вопросы:

1. Что такое Hadoop и какова его основная цель?
2. Что представляет собой Hadoop Distributed File System (HDFS)?
3. Какие основные компоненты входят в экосистему Hadoop?
4. Что такое MapReduce в контексте Hadoop?
5. Какие языки запросов поддерживаются в Apache Hive?
6. Какой инструмент в Hadoop используется для переноса данных между Hadoop и реляционными базами данных?
7. Какова роль HBase в экосистеме Hadoop?
8. Каким образом обеспечивается отказоустойчивость в Hadoop?
9. Какие основные преимущества использования Apache Spark вместо классического MapReduce?
10. Как происходит обмен данными между различными стадиями выполнения MapReduce задачи?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 17

Тема практического занятия: Разработка приложений MapReduce в среде Hadoop

Цель практического занятия: изучение технологии MapReduce, способов разработки и оптимизации приложений для обработки больших данных

В результате выполнения данной работы обучающийся должен
уметь:

- работать с структурированными и неструктурированными данными в контексте MapReduce;
- читать и записывать данные в HDFS из MapReduce приложений;
- работать с ключами и значениями в контексте MapReduce
- отлаживать MapReduce приложения с использованием инструментов, таких как логи и Counters

знать:

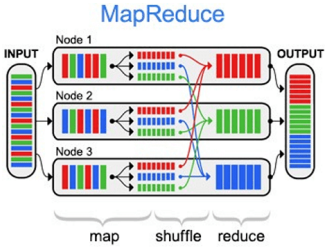
- архитектуру Hadoop, включая HDFS и MapReduce;
- язык программирования для разработки MapReduce приложения в Hadoop;
- API MapReduce, предоставляемого Hadoop, для разработки MapReduce задач
- блочную структуру HDFS и распределения данных;
- способы сортировки и перемешивания данных в фазе Shuffle

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

MapReduce



MapReduce – это модель распределённых вычислений от компании Google, используемая в технологиях Big Data для параллельных вычислений над очень большими (до нескольких петабайт) наборами данных в компьютерных кластерах, и фреймворк для вычисления распределенных задач на узлах (node) кластера

1.10 НАЗНАЧЕНИЕ И ОБЛАСТИ ПРИМЕНЕНИЯ

MapReduce можно по праву назвать главной технологией Big Data, т.к. она изначально ориентирована на параллельные вычисления в распределенных кластерах. Суть MapReduce состоит в разделении информационного массива на части, параллельной обработки каждой части на отдельном узле и финального объединения всех результатов.

Программы, использующие MapReduce, автоматически распараллеливаются и исполняются на распределенных узлах кластера, при этом исполнительная система сама заботится о деталях реализации (разбиение входных данных на части, разделение задач по узлам кластера, обработка сбоев и сообщение между распределенными компьютерами). Благодаря этому программисты могут легко и эффективно использовать ресурсы распределённых Big Data систем.

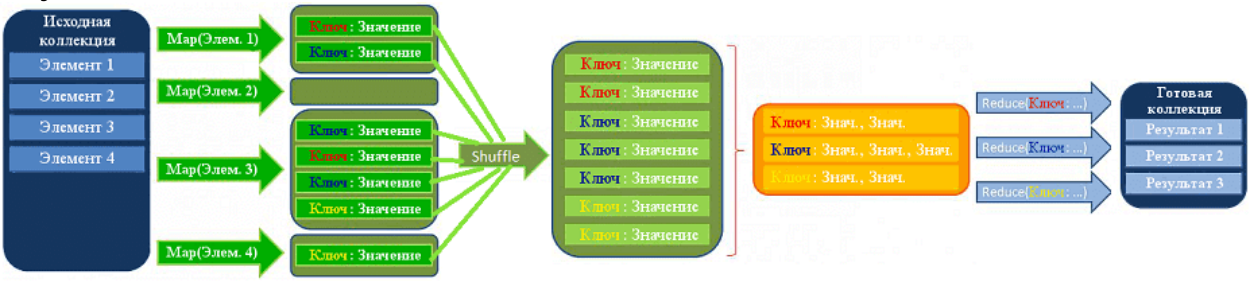
Технология практически универсальна: она может использоваться для индексации веб-контента, подсчета слов в большом файле, счётчиков частоты обращений к заданному адресу, вычисления объём всех веб-страниц с каждого URL-адреса конкретного хост-узла, создания списка всех адресов с необходимыми данными и прочих задач обработки огромных массивов распределенной информации. Также к областям применения MapReduce относится распределённый поиск и сортировка данных, обращение графа веб-ссылок, обработка статистики логов сети, построение инвертированных индексов, кластеризация документов, машинное обучение и статистический машинный перевод. Также MapReduce адаптирована под многопроцессорные системы, добровольные вычислительные, динамические облачные и мобильные среды

1.11 ИСТОРИЯ РАЗВИТИЯ ГЛАВНОЙ ТЕХНОЛОГИИ BIG DATA

Авторами этой вычислительной модели считаются сотрудники Google Джеффри Дин (Jeffrey Dean) и Санджай Гемават (Sanjay Ghemawat), взявшие за основу две процедуры функционального программирования: **map**, применяющая нужную функцию к каждому элементу списка, и **reduce**, объединяющая результаты работы map. В процессе вычисления множество входных пар ключ/значение преобразуется в множество выходных пар ключ/значение

Изначально название MapReduce было запатентовано корпорацией Google, но по мере развития технологий Big Data стало общим понятием мира больших данных. Сегодня множество различных коммерческих, так и свободных продуктов, использующих эту модель распределенных вычислений: Apache Hadoop, Apache CouchDB, MongoDB, MySpace Qizmt и прочие Big Data фреймворки и библиотеки, написанные на разных языках программирования. Среди других наиболее известных реализаций MapReduce стоит отметить следующие:

- Greenplum — коммерческая реализация с поддержкой языков Python, Perl, SQL и пр.;
- GridGain — бесплатная реализация с открытым исходным кодом на языке Java;
- Phoenix — реализация на языке C с использованием разделяемой памяти;
- MapReduce реализована в графических процессорах NVIDIA с использованием CUDA;
- Qt Concurrent — упрощённая версия фреймворка, реализованная на C++, для распределения задачи между несколькими ядрами одного компьютера;
- CouchDB использует MapReduce для определения представлений поверх распределённых документов;
- Skynet — реализация с открытым исходным кодом на языке Ruby;
- Disco — реализация от компании Nokia, ядро которой написано на языке Erlang, а приложения можно разрабатывать на Python;
- Hive framework — надстройка с открытым исходным кодом от Facebook, позволяющая комбинировать подход MapReduce и доступ к данным на SQL-подобном языке;
- Qizmt — реализация с открытым исходным кодом от MySpace, написанная на C#;
- DryadLINQ — реализация от Microsoft Research на основе PLINQ и Dryad.



MapReduce — это разделение, параллельная обработка и свертка распределенных результатов

1.12 КАК УСТРОЕН MAPREDUCE: ПРИНЦИП РАБОТЫ

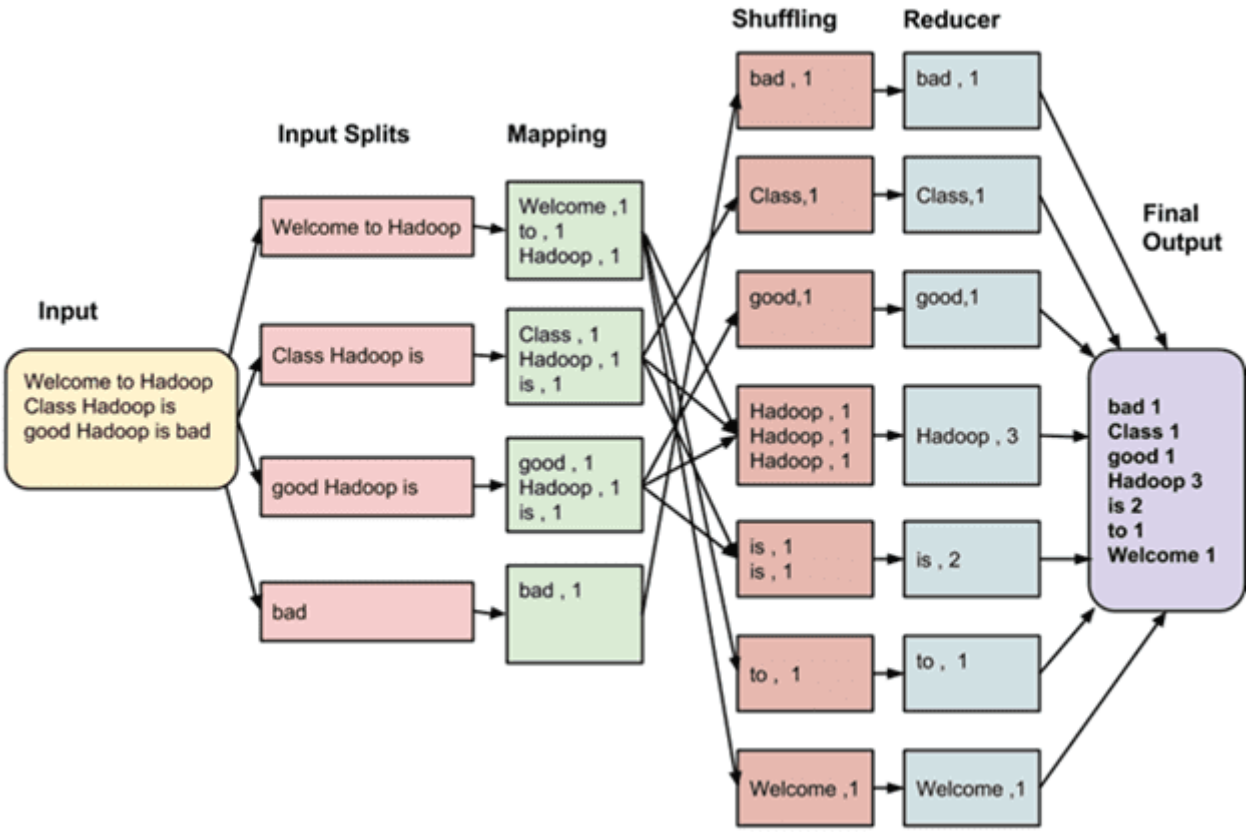
Прежде всего, еще раз поясним смысл основополагающих функций вычислительной модели:

- **Map** принимает на вход список значений и некую функцию, которую затем применяет к каждому элементу списка и возвращает новый список;
- **reduce** (свёртка) — преобразует список к единственному атомарному значению при помощи заданной функции, которой на каждой итерации передаются новый элемент списка и промежуточный результат.

Для обработки данных в соответствии с вычислительной моделью MapReduce следует определить обе эти функции, указать имена входных и выходных файлов, а также параметры обработки.

Сама вычислительная модель состоит из 3-хшаговой комбинации вышеприведенных функций:

1) **Map** – предварительная обработка входных данных в виде большого список значений. При этом главный узел кластера (master node) получает этот список, делит его на части и передает рабочим узлам (worker node). Далее каждый рабочий узел применяет функцию Map к локальным данным и записывает результат в формате «ключ-значение» во временное хранилище.
2) **Shuffle**, когда рабочие узлы перераспределяют данные на основе ключей, ранее созданных функцией Map, таким образом, чтобы все данные одного ключа лежали на одном рабочем узле.
3) **Reduce** – параллельная обработка каждым рабочим узлом каждой группы данных по порядку следования ключей и «склейка» результатов на master node. Главный узел получает промежуточные ответы от рабочих узлов и передаёт их на свободные узлы для выполнения следующего шага. Получившийся после прохождения всех необходимых шагов результат – это и есть решение исходной задачи.



Принцип работы MapReduce

Задание:

Написать map-reduce программу для анализа логов приложения. Алгоритм реализовать на языке Python с помощью библиотеки mrjob. Собрать информацию о количестве логах каждого уровня в системе и провести визуализацию собранной информации.

Технология выполнения работы:

1. На языке Python с помощью модуля logging сгенерируйте файл с большим количеством логов различных уровней (info, error, debug и т.п.). Загрузите этот файл в Hadoop HDFS.
2. С помощью модуля mrjob напишите алгоритм map-reduce, который посчитает количество логов каждого уровня
3. Запустите программу в Hadoop. Результаты работы программы сохраните в файле.
4. Постройте график, на котором показано количество логов каждого уровня и прикрепите его в отчет. Также покажите преподавателю код вашего решения и исходный файл с логами.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Написана программа map – reduce;
- Визуализирована работа программы;
- Создан итоговый отчет

Контрольные вопросы:

1. Что такое MapReduce и какова его основная концепция?
2. Какие основные компоненты составляют программу MapReduce?
3. Что делает Map-функция в MapReduce?
4. Какая роль Reduce-функции в MapReduce?
5. Что представляет собой InputFormat в MapReduce?
6. Какова роль OutputFormat в MapReduce?
7. Какие основные шаги необходимы для написания программы MapReduce?
8. Что такое Combiner в MapReduce и в чем его роль?
9. Как происходит сортировка и группировка данных в MapReduce задаче?
10. Как обеспечивается отказоустойчивость в MapReduce?
11. Какие файлы конфигурации используются в MapReduce?
12. Какие языки программирования можно использовать для разработки MapReduce приложений?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 18

Тема практического занятия: Реализация распределённой обработки неструктурированных и слабоструктурированных данных с Apache Spark

Цель практического занятия: изучение принципов работы Apache Spark и его роли в обработке больших данных

В результате выполнения данной работы обучающийся должен уметь:

- программировать на языках программирования (Scala, Python или Java), которые поддерживаются в Apache Spark;
- работать с DataFrame API для обработки структурированных данных в Spark;
- обрабатывать и анализировать неструктурированные данные, такие как текстовые файлы, с использованием Spark;
- использовать MLlib (Machine Learning Library) для разработки моделей машинного обучения;
- анализировать графовые структуры данных;
- интегрировать Spark с различными источниками данных, такими как HDFS, Amazon S3, Apache Kafka и др.

знать:

- основные компоненты Spark, таких как Spark Core, Spark SQL, Spark Streaming, MLlib и GraphX;
- SQL-подобный язык Spark SQL для работы с данными;
- MLlib (Machine Learning Library) для разработки моделей машинного обучения;
- GraphX и умение анализировать графовые структуры данных

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Apache Spark — популярный open-source фреймворк для распределённой обработки неструктурированных и слабоструктурированных данных. Он реализован в парадигме in-memory вычислений, то есть

обрабатывает данные в оперативной памяти для ускоренного выполнения некоторых классов задач, в частности машинного обучения.

Фреймворк поддерживает работу на Java, R, Python и Scala.

Как правило выделение ресурсов кластера, за исключением архитектурных особенностей, отличается также спецификой поставленной задачи.

Вопрос планирования обработки данных в целях повышения скорости их подготовки часто выносится на уровень инструментов оркестрации. При отсутствии таких инструментов разработчики прибегают к использованию таймеров и прочих встроенных в целевую платформу (Linux Cron) инструментов для планирования запуска заданий, что не меняет сути самого подхода.

Инструменты оркестрации, такие как Airflow, дают возможность управлять на высоком уровне процессом запуска Spark-решения и отслеживать его выполнение, но не управляют самим процессом на микроуровне.

Поэтому в материале будет рассмотрен подход к микроуправлению отдельными заданиями на Spark средствами распараллеливания вычислений, применение инструментов для управления процессами, а также технические составляющие, связанные с параллельным запуском заданий (jobs) и его особенностями.

Изложенное является компиляцией опыта автора в области анализа методов повышения производительности решений на Spark Framework в версиях выше 2.2.0.

Постановка задачи и очевидное решение

Допустим, у нас есть два набора данных (дата-сета). В первом – содержится информация о фильмах и их жанрах, во втором – оценки пользователей. Путём выполнения действий агрегации, объединений и фильтрации мы хотим получить на выходе соответствие идентификатору фильма, количество жанров для него и среднюю оценку пользователей.

Результаты промежуточных вычислений должны быть сохранены на диск для дальнейшего использования. В первую очередь решим задачу самым очевидным способом, а именно разработаем программу с последовательной обработкой данных. Исходя из постановки задачи, необходимо выполнить отдельную обработку входящих наборов данных и их слияние.

Логически решение состоит из следующих последовательно выполняемых функций:

- processGenres – выполняет подготовку данных по жанрам для фильмов;
- processRatings – выполняет вычисление средней пользовательской оценки для фильма;
- processStatistics – выполняет формирование результирующего набора данных.

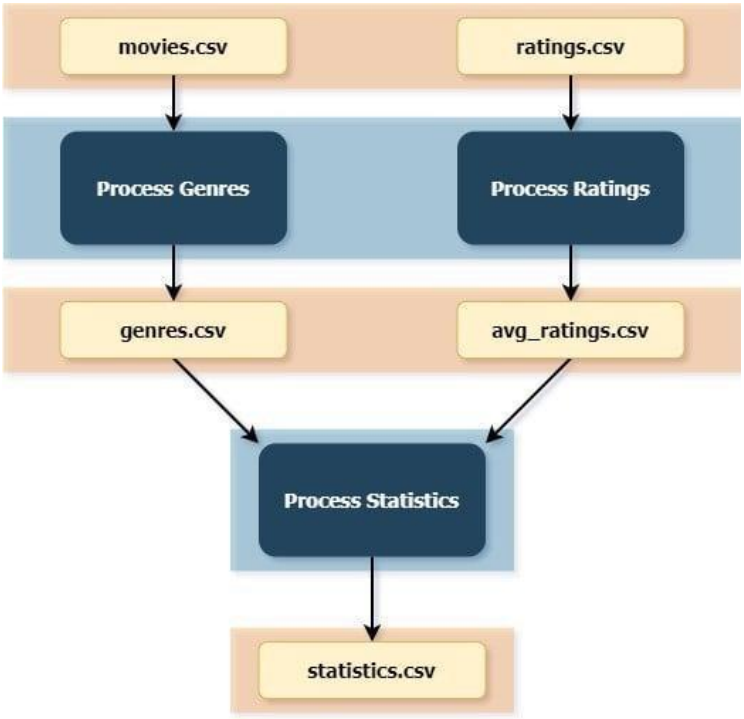


Рисунок 1 Схематичное представление логики приложения

Рисунок 2 Порядок выполнения заданий на временной шкале

Job Id (Job Group)	Description
4	csv at SingleThreading.scala:20 csv at SingleThreading.scala:20
3 (6e32d1a0-3694-4497-b148-226bf6ae3c65)	broadcast exchange (runId 6e32d1a0-3694-4497-b148-226bf6ae3c65) $anonfun$withThreadLocalCaptured$1 at FutureTask.java:264
2	csv at Common.scala:58 csv at Common.scala:58
1	csv at Common.scala:58 csv at Common.scala:58
0	csv at Common.scala:27 csv at Common.scala:27

Рисунок 3 Порядок запуска заданий в списке заданий

```
def main(args: Array[String]): Unit = { val props = ArgProperties(args)
val spark = SparkSession.builder().getOrCreate()
val common = new Common(spark) // выполнение Process Genres
val genresDF = common.processGenres(props.moviesPath, props.genresPath) // выполнение Process Ratings
val avgRatingsDF = common.processRatings(props.ratingsPath, props.avgRatingsPath) // выполнение Process Statistics
common.processStatistics(genresDF, avgRatingsDF, props.statisticsPath)
}
```

Рисунок 4 Код запуска последовательного вычисления

Как видно из приведённого примера, мы получили решение многоэтапной обработки и подготовки данных. Однако очевидно, что оно не

эффективно: задание выполняется достаточно продолжительное время ввиду последовательной обработки данных.

Логика обработки данных инкапуслирована в классе Common. Методы класса Common такие как processGenres и processRatings принимают на вход пути к целевым источникам данных и пути для сохранения результата их преобразования, а также возвращают в качестве результата объект Spark DataFrame результата вычисления.

Метод ProcessStatistics принимает на вход два DataFrame объекта и путь к результирующему набору данных. А класс ArgProperties представляет собой модель параметров командной строки.

Решение такой задачи с независимым запуском отдельных заданий дало бы преимущество и ускорило бы процесс обработки данных. Рассмотрим, каким образом можно выполнить задачу в оптимальном ключе.

Решение с использованием параллельно выполняемых заданий

Spark по умолчанию предоставляет контекст выполнения и логично предположить, что мы могли бы запустить несколько контекстов в рамках нашего вычисления.

Но исходя из рекомендаций разработчиков Spark делать это крайне не рекомендуется, разве что в тестовых целях. Мы можем создать несколько сессий и попробовать выполнить вычисления в них. В любом случае без использования параллельных потоков сделать это не получится.

Spark потокобезопасен для запуска нескольких заданий и в работе с несколькими сессиями. Однако в конфигурировании отдельных сессий для каждого из заданий есть смысл только в том случае, если используются разные конфигурации подключения к источникам данных.

В нашем же случае мы храним данные в одной области, допустим в некотором дисковом пространстве. Тогда будет достаточно создать отдельные задания и выполнить их запуск, например, с помощью механизмов распараллеливания на Python или Scala.

Приведём фрагмент кода из решения, использующий асинхронный запуск заданий на Scala:

```
def main(args: Array[String]): Unit = {
val props = ArgProperties(args)
val spark = SparkSession.builder().getOrCreate()
val common = new Common(spark) // определяем пул потоков, в данном случае размерностью 2
val numThreads = 2 implicit val ec: ExecutionContextExecutorService = ExecutionContext.fromExecutorService(Executors.newFixed
}
```

Рисунок 5 Код запуска решения с асинхронно запускаемыми заданиями

Это решение использует концепцию языка Scala для реализации асинхронных вычислений, концепция реализована с помощью объекта Future. Объект Future инкапсулирует в себе некоторое действие и выполняет его не блокирующим способом, предоставляя интерфейс ожидания и возврата результата.

В данном случае формируется коллекция из двух Future объектов с последующим ожиданием результата из них в основном потоке выполнения программы. Вызов метода sequence от объекта Future возвращает объект Future

инкапсулирующий в себе другие асинхронные действия, в данном случае их два.

В свою очередь Await.result позволяет явно получить результаты из асинхронной операции с указанием времени ожидания, в данном случае бесконечного.

Также определяется пул потоков ExecutionContext ограничивающий количество одновременно выполняемых асинхронных операция, в данном случае размерность пула 2.

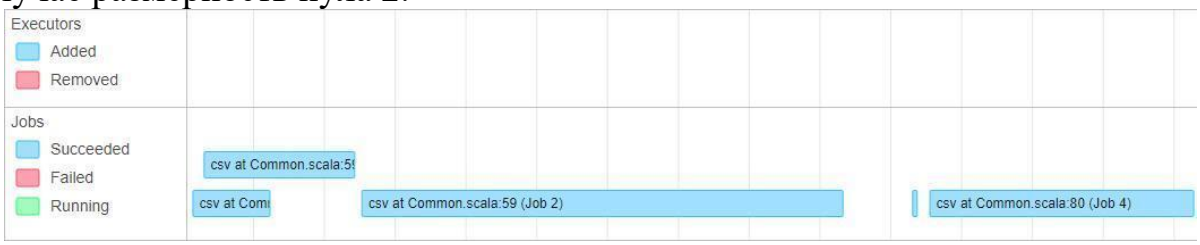


Рисунок 6 Порядок выполнения заданий при асинхронном запуске

Job Id (Job Group)	Description
4	csv at Common.scala:80 csv at Common.scala:80
3 (7cf2f5dc-c3f5-4197-8f5f-df694b56b032)	broadcast exchange (runId 7cf2f5dc-c3f5-4197-8f5f-df694b56b032) $anonfun$withThreadLocalCaptured$1 at FutureTask.java:264
2	csv at Common.scala:59 csv at Common.scala:59
1	csv at Common.scala:59 csv at Common.scala:59
0	csv at Common.scala:28 csv at Common.scala:28

Рисунок 7 Порядок запуска приложений

Как видно на шкале выполнения, теперь задания запускаются асинхронно (задание 1 и задание 2), что ускоряет процесс обработки данных. Запуск происходит с некоторой задержкой, но выполнение происходит параллельно, что видно на рисунки 4.

Конечно, можно было бы решить данную задачу и через стандартные объекты Thread/Runnable предоставляемые платформой Java. Но тогда пришлось бы дополнительно описать логику сохранения промежуточных данных во внешнюю коллекцию, что делает код менее лаконичным и более объёмным.

Очевидный плюс такого решения, кроме повышения скорости работы приложения, заключается в повышенном микроконтроле, когда мы можем отслеживать выполнение заданий в рамках одной сессии, а также более тонко управлять всем процессом обработки данных.

Таким образом, мы получим параллельное выполнение наших задач на кластере из одного приложения на Spark с минимальными изменениями в коде.

Задание:

Создать таблицу с данными в базе данных. Прочитать ее в датафрейм с помощью PySpark. Обработать датафрейм с помощью PySpark.

Визуализировать данные в датафрейме. Подготовьте итоговый отчет о проделанной работе.

Технология выполнения работы:

1. Создайте большую таблицу с данными о крушении Титаника в любой СУБД
2. Прочитайте датасет с помощью pyspark
3. Сделайте предобработку датасета
4. Визуализируйте предобработанные данные на предмет выбросов
5. Визуализируйте распределение предобработанных данных
6. Постройте гистограмму
7. Создайте отчет о проделанной работе

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Выполнены запросы к датафрейму с помощью PySpark;
- Создан итоговый отчет

Контрольные вопросы:

1. Что такое Apache Spark и какова его роль в обработке данных?
2. Какова основная абстракция данных в Spark?
3. Что такое RDD (Resilient Distributed Dataset) в Apache Spark?
4. Какие операции можно выполнять над RDD в Apache Spark?
5. Что такое Spark SQL и какова его роль в Apache Spark?
6. Какова роль DataFrame в Spark SQL?
7. Как Spark обрабатывает слабоструктурированные данные, такие как JSON?
8. Что такое GraphX в Apache Spark и для чего он используется?
9. Как Spark обрабатывает неструктурированные данные, такие как текстовые файлы?
10. Какие инструменты и языки программирования можно использовать для взаимодействия с Apache Spark?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 19

Тема практического занятия: Использование полносвязных и сверточных сетей

Цель практического занятия: обеспечение практического опыта работы с полносвязными и сверточными нейронными сетями, понимание их принципов работы и способность применять к реальным задачам

В результате выполнения данной работы обучающийся должен

уметь:

- создавать и обучать полносвязные нейронные сети;
- работать с различными активационными функциями, такими как ReLU, Sigmoid, и Tanh;
- работать с сверточными слоями, пулингом (pooling), и слоями объединения (concatenation)
- использовать сверточные сети для задач компьютерного зрения, таких как классификация изображений и сегментация объектов

знать:

- базовые концепции нейронных сетей, таких как нейроны, веса, активационные функции и передача сигнала;
- понятия и принцип работы слоев, включая входной, скрытый и выходной слои
- функции потерь и способы выбора подходящей функции в зависимости от задачи;
- базовую структуру сверточных нейронных сетей

Перечень оборудования, необходимого для выполнения задания:

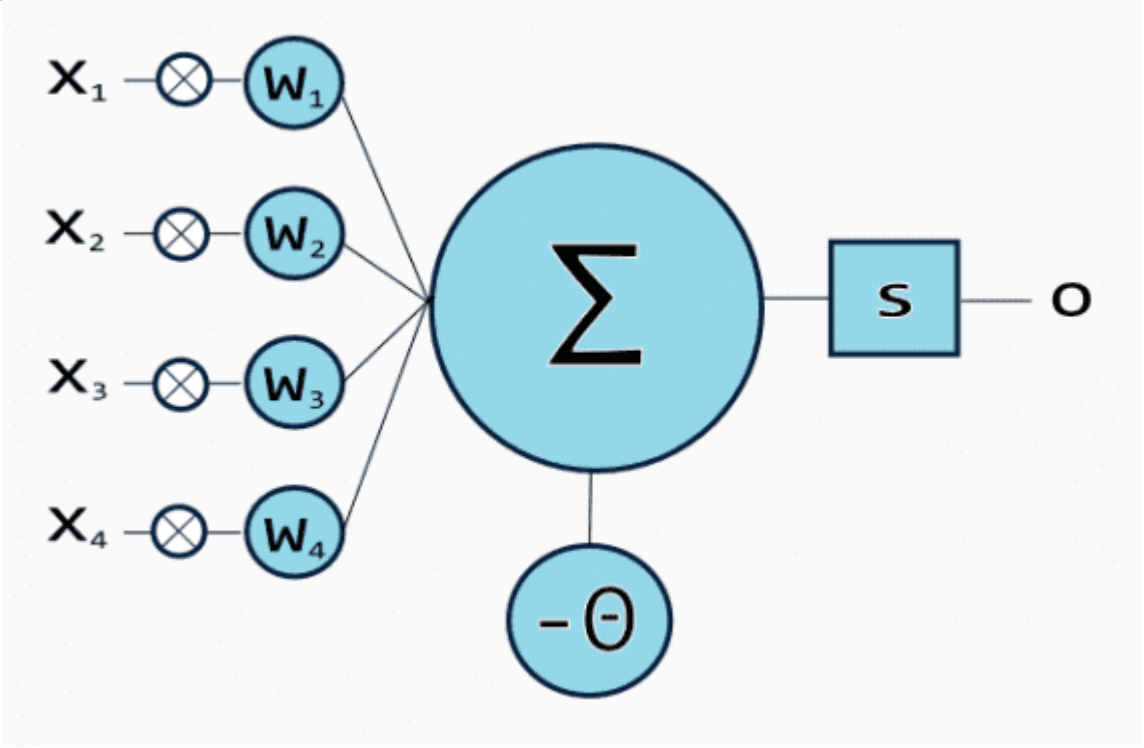
- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Искусственная нейронная сеть **(ИНС)** — вычислительная нелинейная модель, в основе которой лежит нейронная структура мозга, способная обучаться выполнению задач классификации, предсказания, принятия решений, визуализации и некоторых других только благодаря рассмотрению примеров.

Любая архитектура ИНС состоит из искусственных нейронов — элементов обработки, имеющих структуру 3 связанных друг с другом слоев: входным, состоящим из одного или более слоев скрытым и выходным.

Входной слой состоит из входных нейронов, которые передают информацию в скрытый слов. Скрытый слой в свою очередь передает информацию в выходной. Каждый нейрон имеет входы с весами — синапсами, функцию активации, определяющую выходную информацию при заданной входной, и один выход. Синапсы — регулируемые параметры, конвертирующие нейронную сеть в параметризованную систему.



Искусственная нейронная сеть с 4 входами

Функция активации

Взвешенная сумма со входов — активационный сигнал — проходит через функцию активации для вывода данных из нейрона. Есть несколько видов функции активации: линейная, ступенчатая, сигмоидная, тангенциальная, выпрямительная (Rectified linear unit, ReLu).

Линейная функция

$f(x)=ax$

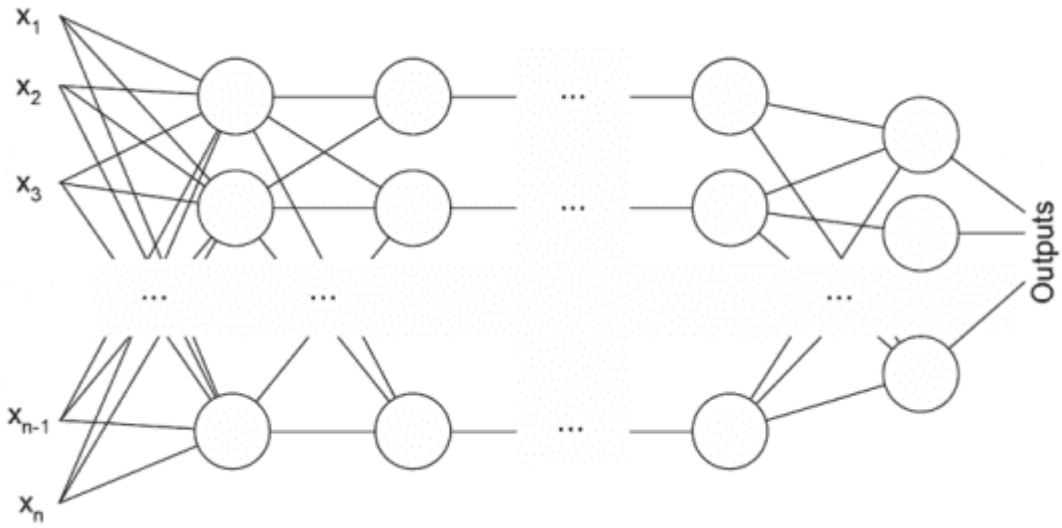
Обучение (или тренировка) — процесс оптимизации весов, в котором минимизируется ошибка предсказания, и сеть достигает требуемого уровня точности. Наиболее используемый метод для определения вклада в ошибку каждого нейрона — **обратное распространение ошибки**, с помощью

которого вычисляют градиент. Это одна из модификаций метода градиентного спуска.

С помощью дополнительных скрытых слоев возможно сделать систему более гибкой и мощной. ИНС с многими скрытыми слоями называются глубокими нейронными сетями (deep neural network, DNN); они создают сложные нелинейные связи.

Рассмотрим популярные архитектуры нейронных сетей, которые хорошо показали себя в задачах NLP и рекомендуются к использованию.

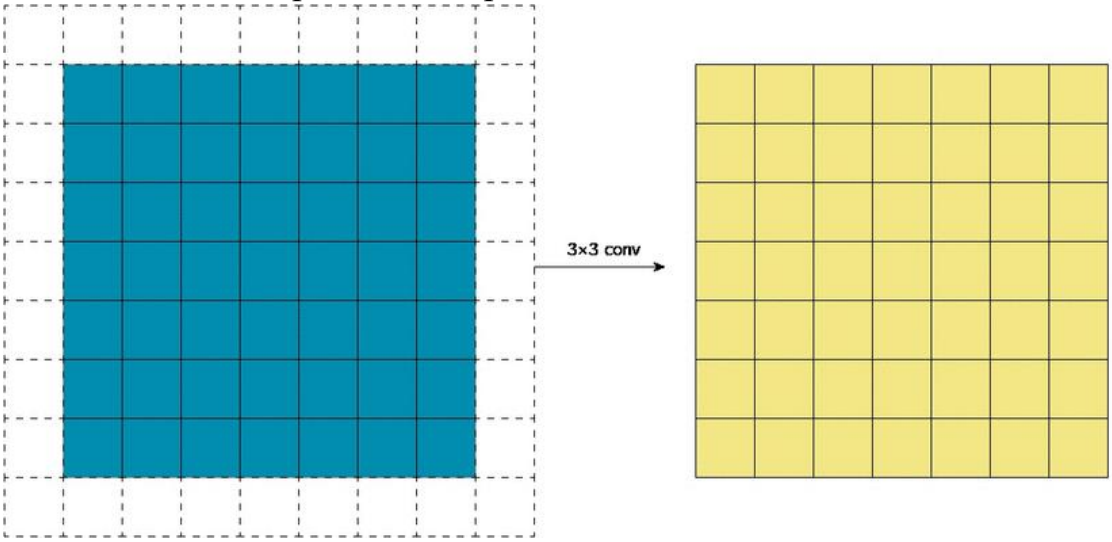
1.13 1. Многослойный перцептрон



Перцептрон

Многослойный перцептрон состоит из 3 или более слоев. Он использует нелинейную функцию активации , часто тангенциальную или логистическую, которая позволяет классифицировать линейно неразделимые данные. Каждый узел в слое соединен с каждый узлом в последующем слое, что делает сеть полностью связанной. Такая архитектура находит применение в задачах распознавания речи и машинном переводе.

1.14 2. Сверточная нейронная сеть



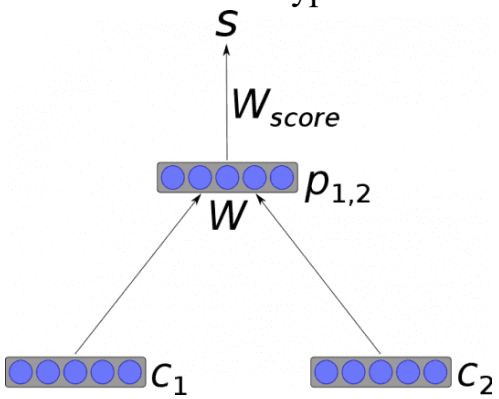
Сверточная нейронная сеть (Convolutional neural network, CNN) содержит один или более объединенных или соединенных сверточных слоев. CNN использует вариацию многослойного перцептрона, рассмотренного

выше. Сверточные слои используют операцию свертки для входных данных и передают результат в следующий слой. Эта операция позволяет сети быть глубже с меньшим количеством параметров.

Сверточные сети показывают выдающиеся результаты в приложениях к картинкам и речи. В статье Convolutional Neural Networks for Sentence Classification автор описывает процесс и результаты задач классификации текста с помощью CNN. В работе представлена модель на основе word2vec, которая проводит эксперименты, тестируется на нескольких бенчмарках и демонстрирует блестящие результаты.

В работе Text Understanding from Scratch авторы показывают, что сверточная сеть достигает выдающихся результатов даже без знания слов, фраз предложений и любых других синтаксических или семантических структур присущих человеческому языку. Семантический разбор, поиск парафраз, распознавание речи — тоже приложения CNN.

1.15 3. Рекурсивная нейронная сеть



Рекурсивная нейронная сеть — тип глубокой нейронной сети, сформированный при применении одних и тех же наборов весов рекурсивно над структурой, чтобы сделать скалярное или структурированное предсказание над входной структурой переменного размера через активацию структуры в топологическом порядке. В простейшей архитектуре нелинейность, такая как тангенциальная функция активации, и матрица весов, разделяемая всей сетью, используются для объединения узлов в родительские объекты.

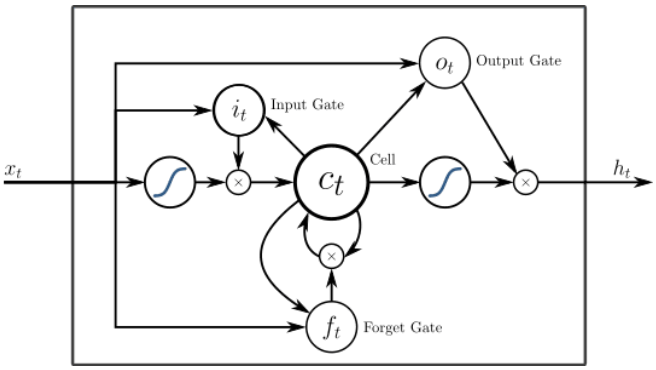
1.16 4. Рекуррентная нейронная сеть

Рекуррентная нейронная сеть, в отличие от прямой нейронной сети, является вариантом рекурсивной ИНС, в которой связи между нейронами — направленные циклы. Последнее означает, что выходная информация зависит не только от текущего входа, но также от состояний нейрона на предыдущем шаге. Такая память позволяет пользователям решать задачи NLP: распознание рукописного текста или речи. В статье Natural Language Generation, Paraphrasing and Summarization of User Reviews with Recurrent Neural Networks авторы показывают модель рекуррентной сети, которая генерирует новые предложения и краткое содержание текстового документа.

Siwei Lai, Liheng Xu, Kang Liu, и Jun Zhao в своей работе Recurrent Convolutional Neural Networks for Text Classification создали рекуррентную

сверточную нейросеть для классификации текста без рукотворных признаков. Модель сравнивается с существующими методами классификации текста — Bag of Words, Bigrams + LR, SVM, LDA, Tree Kernels, рекурсивными и сверточными сетями. Описанная модель превосходит по качеству традиционные методы для всех используемых датасетов.

1.17 5. LSTM



LSTM блок с входным, выходным и гейтом забывания

Сеть долгой краткосрочной памяти (Long Short-Term Memory, LSTM) — разновидность архитектуры рекуррентной нейросети, созданная для более точного моделирования временных последовательностей и их долгосрочных зависимостей, чем традиционная рекуррентная сеть. LSTM-сеть не использует функцию активации в рекуррентных компонентах, сохраненные значения не модифицируются, а градиент не стремится исчезнуть во время тренировки. Часто LSTM применяется в блоках по несколько элементов. Эти блоки состоят из 3 или 4 затворов (например, входного, выходного и гейта забывания), которые контролируют построение информационного потока по логистической функции.

В Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling авторы показывают архитектуру глубокой LSTM рекуррентной сети, которая достигает хороших результатов для крупномасштабного акустического моделирования.

В работе Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network представлена модель для автоматической морфологической разметки. Модель показывает точность 97.4 % в задаче разметки. Apple, Amazon, Google, Microsoft и другие компании внедрили в продукты LSTM-сети как фундаментальный элемент.

1.18 6. Sequence-to-sequence модель

Часто Sequence-to-sequence модели состоят из двух рекуррентных сетей: кодировщика, который обрабатывает входные данные, и декодера, который осуществляет вывод.

Sequence-to-Sequence модели часто используются в вопросно-ответных системах, чат-ботах и машинном переводе. Такие многослойные ячейки успешно использовались в sequence-to-sequence моделях для перевода в статье Sequence to Sequence Learning with Neural Networks study.

В Paraphrase Detection Using Recursive Autoencoder представлена новая рекурсивная архитектура автокодировщика, в которой представления —

вектора в n-мерном семантическом пространстве, где фразы с похожими значением близки друг к другу.

7. Неглубокие (shallow) нейронные сети

Неглубокие модели, как и глубокие нейронные сети, тоже популярные и полезные инструменты. Например, word2vec — группа неглубоких двухслойных моделей, которая используется для создания векторных представлений слов (word embeddings). Представленная в Efficient Estimation of Word Representations in Vector Space, word2vec принимает на входе большой корпус текста и создает векторное пространство. Каждому слову в этом корпусе приписывается соответствующий вектор в этом пространстве. Отличительное свойство — слова из общих текстов в корпусе расположены близко друг к другу в векторном пространстве.

В статье описаны архитектуры нейронных сетей: глубокий многослойный перцептрон, сверточная, рекурсивная, рекуррентная сети, нейросети долгой краткосрочной памяти, sequence-to-sequence модели и неглубокие (shallow) сети, word2vec для векторных представлений слов. Кроме того, было показано, как функционируют эти сети, и как различные модели справляются с задачами обработки естественного языка. Также отмечено, что сверточные нейронные сети в основном используются для задач классификации текста, в то время как рекуррентные сети хорошо работают с воспроизведением естественного языка или машинным переводом. В следующих части серии будут описаны существующие инструменты и библиотеки для реализации описанных типов нейросетей.

Задание:

Написать полносвязную и сверточную нейросети. Подготовить данные для обучения нейросетей. Провести обучение на подготовленных данных. Сравнить их работу. Подготовить итоговый отчет о проделанной работе.

Технология выполнения работы:

1. Создайте модель для распознавания рукописных цифр из набора MNIST (можно воспользоваться ноутбуком 1-го занятия) и проведите ряд тестов:
2. Запустите сеть с различными размерами обучающей и проверочной выборок:
   - Обучающая выборка 50.000 примеров
   - Обучающая выборка 10.000 примеров
   - Обучающая выборка 500 примеров
3. Создайте еще два варианта сети и сравните значения точности на проверочной выборке (на последней эпохе) и на тестовой выборке. Сделайте сравнительную таблицу. 3. Создайте сеть следующей архитектуры:
   - 4 Dense слоя
   - 3 Dropout слоя
   - 3 BatchNormalization слоя

- Создайте сверточную модель для решения той же задачи
- Сравните результаты работы двух моделей в отчете.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Созданы две нейросети;
- Подготовлен отчет с подробным сравнением работы нейростетей

Контрольные вопросы:

1. Что такое полносвязная нейронная сеть (Fully Connected Neural Network)?
2. Какова основная структура сверточной нейронной сети (Convolutional Neural Network, CNN)?
3. Как работает операция свертки в сверточной нейронной сети?
4. Какую роль выполняют слои подвыборки (Pooling) в CNN?
5. Что такое активационная функция и зачем она нужна в нейронных сетях?
6. Каким образом применяется dropout в нейронных сетях и для чего он используется?
7. Что такое функция потерь (Loss Function) и как она используется при обучении нейронных сетей?
8. Какую роль выполняет функция активации ReLU (Rectified Linear Unit) в нейронных сетях?
9. Какие типы слоев могут использоваться в сверточных нейронных сетях, помимо сверточных и подвыборки?
10. Что такое батч-нормализация (Batch Normalization) и для чего она используется в нейронных сетях?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. 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.

Практическое занятие № 20

Тема практического занятия: Рекуррентные и одномерные свёрточные сети для обработки текстовой информации

Цель практического занятия: приобретение теоретических и практических навыков по использованию современных методов глубокого обучения для обработки текстовой информации.

В результате выполнения данной работы обучающийся должен уметь:

- подготавливать и обрабатывать текстовые данные для входа в RNN;
- использовать RNN для задач обработки текста, таких как предсказание следующего слова или классификация текста;
- проводить обучение и дообучение нейростетей для решения задач;
- оценивать качество модели

знать:

- основные принципы работы RNN;
- применение RNN в обработке текста;
- преимуществ и недостатков рекуррентных сетей при обработке текста

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

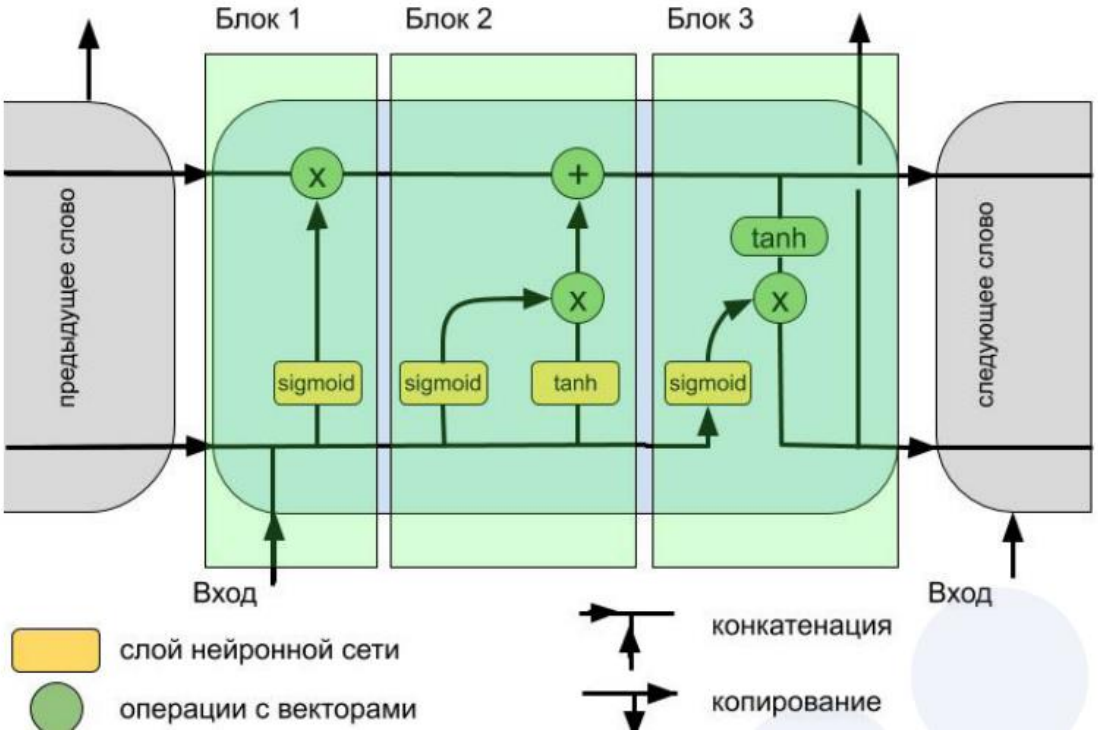
Общие теоретические сведения

Рекуррентные сети Рекуррентные сети (RNN) — это сети, у которых есть «память», учитывающая предшествующую информацию. Рассмотрим пример: у нас есть видео какого-то объекта, например, летящего мяча. Видео разбито на кадры, и нам нужно понять, в какую сторону летит мяч. Если мы посмотрим на какой-нибудь кадр, то увидим следующую картину: Мяч просто находится в какой-то точке. Если посмотрим другой кадр, то увидим, что мяч тоже будет находиться в какой-то точке. Чтобы понять, в какую сторону двигается мач, нам нужно оценить несколько кадров, то есть посмотреть положение мяча, например, на трех последовательных кадрах: Глядя на предыдущие кадры, понимаем, что шар двигается влево. Рекуррентные сети основаны на том, что делают вычисления на основе нынешних данных с учетом предыдущих. Отсюда следует определение. Рекуррентные нейронные сети — вид

нейронных сетей, где связи между элементами образуют направленную последовательность. Теперь рассмотрим, как они устроены, на примере одного нейрона. Допустим, нам нужно предсказать слово в предложении «Я иду ...». Наш нейрон принимает на вход первое слово, обрабатывает его и подаёт на выход, запоминает, и при обработке следующего слова также подаст на вход выход обработки первого слова («я»). Обработав информацию с учетом первого и второго слова, нейрон что-то подаёт на выход и запоминает этот выход («я иду»), чтобы подать его на вход нашего искомого слова. Дальше алгоритм подбирает слово «гулять» по смыслу. И всё на основе последовательных связей. Рекуррентные сети это всё выполняет один нейрон! Он создаёт как бы свою копию («виртуальный» нейрон) Благодаря этому мы можем получить выход не только из последнего «виртуального» нейрона, но и на любом шаге. Главная проблема такого подхода — это проблема исчезающего градиента. Проблема исчезающего градиента — это проблема потери информации. Чем длиннее последовательность, тем больше вероятность того, что информация о первых значениях будет теряться. Для предсказания слов в коротких последовательностях, например, «я иду ...», «поэт Александр Сергеевич ...» и т. д. такой проблемы не возникает. Но если мы будем иметь дело с текстом: «В детстве я несколько лет провел во Франции. ... ... Я хорошо говорю по …», то мысль о Франции потеряется, и подбор правильного слова по смыслу «Я хорошо говорю по-французски» маловероятен. Решает эту проблему рекуррентный слой LSTM. Рекуррентные сети Сети LSTM LSTM (long short-term memory), дословно «долгая краткосрочная память» — тип рекуррентной нейронной сети, способной обучаться долгосрочным зависимостям. LSTM специально разработаны для устранения проблемы исчезающего градиента. Их специализация — запоминание информации в течение длительных периодов времени. Рассмотрим принцип работы LSTM сетей. Рекуррентные сети имеют форму цепочки повторяющихся «виртуальных» нейронов. В каждом таком нейроне происходят настраиваемые преобразования. Структуры, выполняющие эти преобразования, называются «гейтами». Они состоят из слоев нейронной сети и операций с векторами (сложение, умножение). Горизонтальная линия в верхней части ячейки — это состояние ячейки. Оно последовательно проходит через определенные блоки и преобразуется за счет операций каждого блока. Ячейка условно разделена на три блока. Блок 1. Слой потери. Информация, пришедшая из предыдущего нейрона (предыдущее слово), в совокупности с входными данными (текущее слово) проходит через сигмоиду и умножается на текущее значение состояния ячейки (оно пришло из предыдущей ячейки). Этот блок определяет, сколько информации из предыдущих ячеек нужно забыть, а сколько пропустить дальше. На выходе слоя sigmoid получаются значения от 0 до 1. 0 — это не пропускать ничего, 1 — это пропустить всё. Пропускается некоторая часть информации (в зависимости от значения). Блок 2. Слой сохранения. Этот блок отвечает за информацию, которая пойдет в следующую ячейку. В слое сохранения происходит фиксация обновленных данных. В sigmoid также определяется, сколько информации нужно забыть, а

сколько пропустить. Слой tanh создает вектор новых значений (перемножаем на значение sigmoid (0,1)), которые добавляются в состояние ячейки. Блок 3. Слой обновления. Этот блок отвечает за то, что пойдёт в следующую ячейку. Пропускаем состояние ячейки через tanh (чтобы разместить все значения в интервале [-1, 1]) и умножаем на выходной сигнал sigmoid. Теперь поговорим о Выходе ячейки. Так как каждая такая ячейка — это «копия» нашего нейрона, то на выходе может быть либо одно значение (результирующая всех таких виртуальных нейронов), либо список выходов каждой ячейки (на рисунке это вертикальная стрелка в правом верхнем углу «Выход») .

Для применения LSTM на практике не обязательно знать принцип его работы.



Рекуррентные сети LSTM слой в Keras В Keras LSTM слой добавляется следующим образом:

```
# Вариант 1
model = Sequential() model.add(Embedding(50, 10, input_length=1000))
model.add(LSTM(32))
```

Слой Embedding имеет размерность 1000x10, а слой LSTM имеет размерность 32. В этом случае слой LSTM на выходе каждого нейрона имеет только одно значение. Если при создании слоя LSTM мы укажем значение параметра return_ sequences=True, то на выходе каждого нейрона будет не одно значение, а несколько (равное размеру предыдущего слоя).

```
# Вариант 2
model = Sequential()
model.add(Embedding(50, 10, input_length=1000))
model.add(LSTM(32, return_sequences=True))
```

В данном случае размерность после слоя LSTM будет 1000x32. Двунаправленные рекуррентные сети Bidirectional Для предсказания слов в текстах довольно удобно применять сети, которые знают, что было ДО

текущего слова. Но еще лучше, если мы будем знать и то, какие слова стоят ПОСЛЕ предсказываемого. Это можно сделать при помощи двунаправленной RNN. Для этого в Keras существует обертка Bidirectional(). Она обрабатывает последовательность не только с начала до конца, но и наоборот.

```
model = Sequential()
# Двунаправленная RNN
model.add(Bidirectional(LSTM(64, return_sequences=True), input_shape=(5, 10))) model.add(Bidirectional(LSTM(32))) model.add(Dense(10))
```

Это та же LSTM, только работающая в две стороны. Рекуррентные сети Сети GRU GRU можно рассматривать как более простую версию сетей LSTM. Он включает в себя много схожих понятий, но имеет меньше параметров, и поэтому при одном и том же размере скрытого слоя GRU обучается быстрее.

```
# Одномерная свертка model = Sequential()
model.add(Embedding(50, 10, input_length=1000))
model.add(GRU(16, return_sequences=True))
```

Так же, как и LSTM, имеет параметр return_sequences. Управляемый рекуррентный нейрон и сети долгой краткосрочной памяти имеют сопоставимую точность, а в некоторых случаях GRU даже точнее (но такое бывает довольно редко). Поэтому, прежде чем использовать LSTM, можно попробовать сделать нейронку на GRU, чтобы сэкономить время. Рекуррентные сети Рассмотрим пример одномерной свертки на сети для обработки текстов. Создавая слой, мы указываем следующие параметры: окно свертки (фильтры) и ядро свертки. Чтобы посчитать все значения свертки, нужно каждый элемент окна умножить на каждый элемент ядра и сложить их. После этого смещаемся на один элемент вправо и повторяем процедуру, и так до тех пор, пока не дойдем до последнего элемента последовательности. Рассмотрим пример на картинке: первое значение свертки будет считаться как N = 46*0.2 + 9*0.2 + 124*0.2 + 56*0.2 = 47. И далее так считаем все значения нашей последовательности. В Keras этот слой применяется следующим образом: # Одномерная свертка model = Sequential() model.add(Embedding(50, 10, input_length=1000)) model.add(Conv1D(20, 5, activation='relu')) При создании слоя Conv1D указывается количество фильтров (filters=20) и размер ядра (kernel_size=5). Преимущества одномерной сверточной нейронной сети: • время обучения значительно ниже, чем у рекуррентных нейронных сетей Недостатки одномерной сверточной нейронной сети: • нет возможности «запомнить» нужные данные на длительный срок • недостаток можно устранить с помощью механизма «внимания»

Задание:

Разработать и обучить нейронную сеть для классификации текстов. Воспользуйтесь набором данных для анализа тональности, содержащим новостные заголовки и их соответствующую тональность. Вы можете использовать публичные датасеты, такие как IMDb Movie Reviews, Twitter Sentiment Analysis, или другие, доступные в открытых источниках.

Технология выполнения работы:

1. Изучение материалов:
   - Ознакомьтесь с принципами работы рекуррентных нейронных сетей (RNN) и одномерных сверточных нейронных сетей (CNN) для обработки текстовой информации.
   - Проведите анализ преимуществ и недостатков использования RNN и CNN в задачах обработки текстов.
2. Выбор датасета:
   - Выберите подходящий датасет для работы с текстовой информацией. Это может быть датасет для задачи классификации текстов, анализа тональности или другой подходящей задачи.
3. Подготовка данных:
   - Загрузите выбранный датасет.
   - Проведите предварительную обработку данных: токенизация, удаление стоп-слов, приведение к нижнему регистру и др.
4. Реализация моделей:
   - Реализуйте рекуррентную нейронную сеть для обработки текста.
   - Реализуйте одномерную сверточную нейронную сеть для обработки текста.
   - Используйте библиотеку, такую как TensorFlow, PyTorch или Keras, для упрощения процесса.
5. Обучение моделей:
   - Разделите данные на обучающую и тестовую выборки.
   - Обучите рекуррентную и одномерную сверточную модели на обучающих данных.
   - Оцените производительность моделей на тестовой выборке.
6. Сравнение результатов:
   - Сравните результаты работы рекуррентной и сверточной моделей.
   - Оцените их производительность по метрикам, таким как точность, полнота, F1-мера.
7. Анализ результатов:
   - Проанализируйте преимущества и недостатки каждой модели.
   - Рассмотрите ситуации, в которых одна модель может быть более предпочтительной.
8. Визуализация:
   - Визуализируйте процесс обучения для каждой модели (график функции потерь по эпохам).
9. Дополнительные эксперименты:
   - Проведите дополнительные эксперименты, изменяя гиперпараметры моделей.
   - Рассмотрите возможность использования предобученных эмбеддингов для улучшения производительности.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Проведено описание выбранного датасета;
- Проведена предварительная обработка данных;
- Обучены рекуррентная и одномерная сверточные модели на обучающих данных.;
- Проведена оценка производительности по выбранным метрикам;
- Визуализирован процесс обучения каждой модели

Контрольные вопросы:

1. Что такое рекуррентная нейронная сеть (RNN) и как она обрабатывает последовательности данных?
2. Какие основные проблемы возникают при использовании простых рекуррентных сетей, и как эти проблемы решаются?
3. Что представляют собой LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit) в контексте рекуррентных нейронных сетей?
4. Что такое одномерная сверточная нейронная сеть (1D CNN) и для чего она используется при обработке текстовых данных?
5. Как работает операция свертки в одномерной сверточной нейронной сети?
6. Каким образом устроен процесс обучения рекуррентных нейронных сетей для обработки текстов?
7. Что такое векторное представление слов (Word Embedding) и как оно используется в текстовых моделях?
8. Как происходит обработка переменной длины последовательностей в рекуррентных нейронных сетях при работе с текстовыми данными?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.

Практическое занятие № 21

Тема практического занятия: Регрессионный анализ. Методы прогнозирования временных рядов

Цель практического занятия: освоить основные методы прогнозирования, включая использование регрессионных моделей, временных рядов и соответствующих инструментов.

В результате выполнения данной работы обучающийся должен уметь:

- подготавливать данные для обучения и тестирования моделей, включая масштабирование, обработку пропущенных значений и кодирование категориальных признаков;
- выбирать подходящие модели для задачи прогнозирования;
- проводить оценку моделей на основе метрик, таких как среднеквадратичная ошибка (MSE) или коэффициент детерминации (R^2)
- Работать с регрессионными моделями и проводить их интерпретацию;
- Использовать рекуррентные нейронные сети (RNN) для анализа последовательных данных;
- применять сверточные нейронные сети (CNN) для анализа временных рядов;

знать:

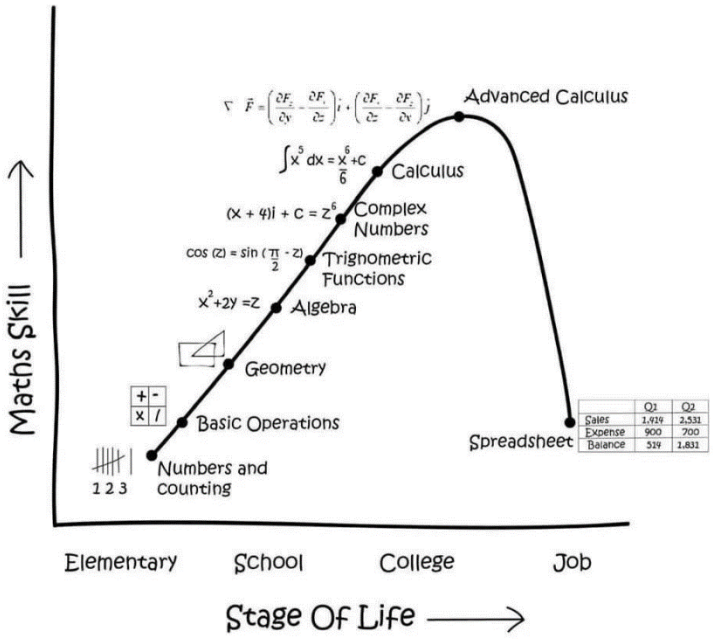
- основы регрессионного анализа;
- основы временных рядов;
- принципы работы и архитектуры искусственных нейронных сетей;
- библиотеки для работы с нейронными сетями, такими как TensorFlow или PyTorch

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Анализ временных рядов



Временной ряд отличается от обычной последовательности тем, что каждое его значение взято на одинаково расположенных точках времени. На обложке статьи показан пример.

Стоит упомянуть, что мы можем найти очень большое количество литературы об алгоритмах анализа временных рядов, потому, что необходимость прогнозировать именно этот тип данных возникает задолго до машинного обучения. В промышленности многие специалисты по данным до сих пор используют простые авторегрессионные модели вместо глубокого обучения для предсказания временных рядов. Обычно основными причинами выбора этих методов остаются интерпретируемость, ограниченность данных, простота использования и стоимость обучения. Поэтому, первая часть статьи посвящена статистике: статистические понятия в анализе временных рядов и классические алгоритмы прогнозирования.

Основные понятия в статистическом анализе временных рядов:

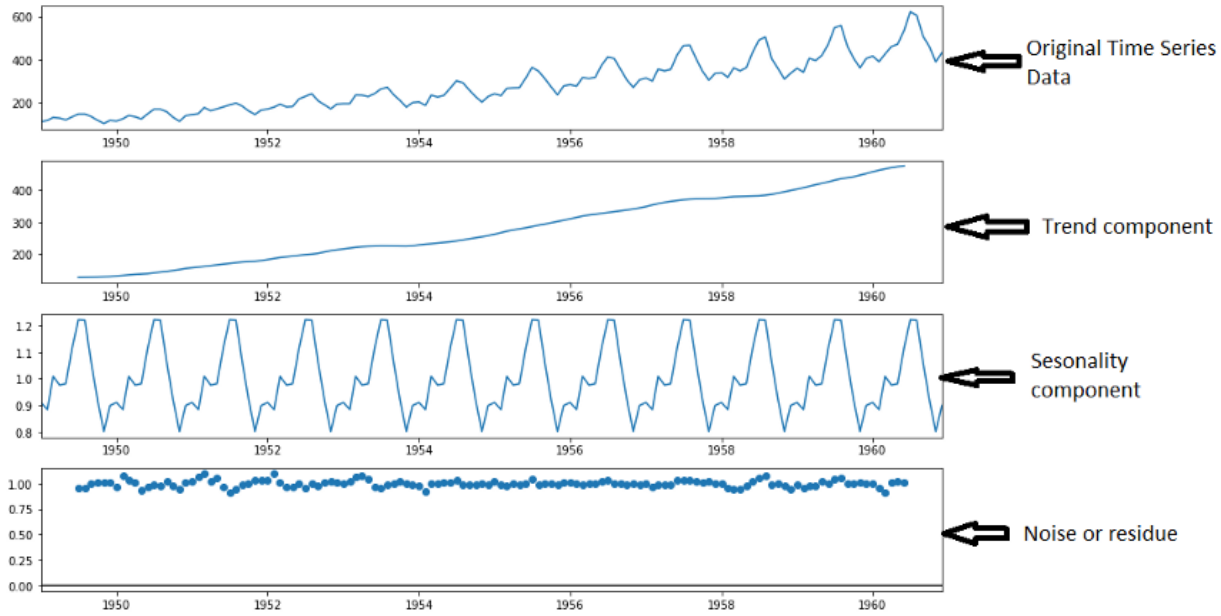
Тренд - компонента, описывающая долгосрочное изменение уровня ряда.

Сезонность - компонента, обозначаемая как Q, описывает циклические изменения уровня ряда.

Ошибка (random noise) - непрогнозируемая случайная компонента, описывает нерегулярные изменения в данных, необъяснимые другими компонентами.

Автокорреляция — статистическая взаимосвязь между последовательностями величин одного ряда. Это один из самых важных коэффициентов в анализе временного ряда. Чтобы посчитать автокорреляцию, используется корреляция между временным рядом и её сдвинутой копией от величины временного сдвига. Сдвиг ряда называется лагом.

Автокорреляционная функция- график автокорреляции при разных лагах



Компоненты ряда

К такому графику, можно добавить также визуализацию автокорреляционной функции.

Стационарный ряд - ряд, в котором свойства не зависят от времени. При таких компонентах ряда, как тренд или сезонность, ряд не является стационарным. Это интуитивное понятие. На практике есть два вида стационарности - строгая и слабая. В следующих алгоритмах, нам достаточно слабой. Она заключается в постоянной дисперсии (без гетероскедастичности) и в постоянности среднего значения ряда.

Гетероскедастичность - не равномерная дисперсия. То есть, не однородность наблюдений.

В python, например, реализованы библиотеки, через которые мы можем проверить ряд на стационарность, гетероскедастичность или построить автокорреляционную функцию.

Авторегрессионная модель (Auto regression method (AR))

Это линейная модель, в которой прогнозированная величина является суммой прошлых значений, умноженных на числовой множитель.

Скользящее среднее (Moving average method (MA))

Скользящее среднее-это расчет для анализа точек данных путем создания ряда средних значений различных подмножеств полного набора данных. Мне этот метод часто напоминает Тренд ряда. Скользящее среднее часто применяется для анализа временных рядов акций.

Метод скользящего среднего с авторегрессией и интегрированием (Auto regressive integrated moving average (ARIMA)

Существует также модель ARMA. Которая представляет собой сочетание моделей выше. Разница между моделями ARMA и ARIMA описана ниже.

Теорема Волда — утверждение математической статистики, согласно которому каждый временной ряд можно представить в виде скользящего среднего бесконечного порядка

Так как модель ARMA , согласно теореме о разложении Волда , теоретически достаточна для описания регулярного **стационарного** временного ряда, мы заинтересованы в стационарности ряда.

Почему стационарность так важна? Предполагается, что, если временной ряд ведет себя определенным образом, очень высока вероятность того, что он повторит те же паттерны в будущем.

Поэтому, нам стоит применить преобразование для нестационарного ряда. Здесь есть два подхода - удаление тренда и взятие разности. Почему тут интегрирование? В математике есть раздел - исчисление конечных разностей, в котором интегрированием называется взятие разности .

Вот такое скромное описание основных моделей прогнозирования, применяемых ещё до машинного обучения. Если вам интересна эта тема, советую почитать об ARIMA с сезонностью (Seasonal auto regressive integrated moving average (SARIMA)) или о моделях ARCH, GARCH

1.19 Нейронные сети для предсказания временных рядов

Давайте рассмотрим, с какими проблемами может столкнуться нейронный алгоритм в работе с временными рядами

Одна из идей, сделавшая RNN неоправданно эффективными - «авторегрессия» (auto-regression), это значит, что созданная переменная добавляется в последовательность в качестве входных данных. В машинном

обучении часто применяется эта техника, особенно в работе с временными рядами.

Хотя рекуррентная сеть и должна работать со всей последовательностью, к сожалению, присутствует проблема «затухающего градиента»(vanishing gradient problem). Что значит, что более старые входы не влияют на текущий выход. Такие модели, как LSTM **пытаются** решить эту проблему, добавляя дополнительные параметры (separate memory).

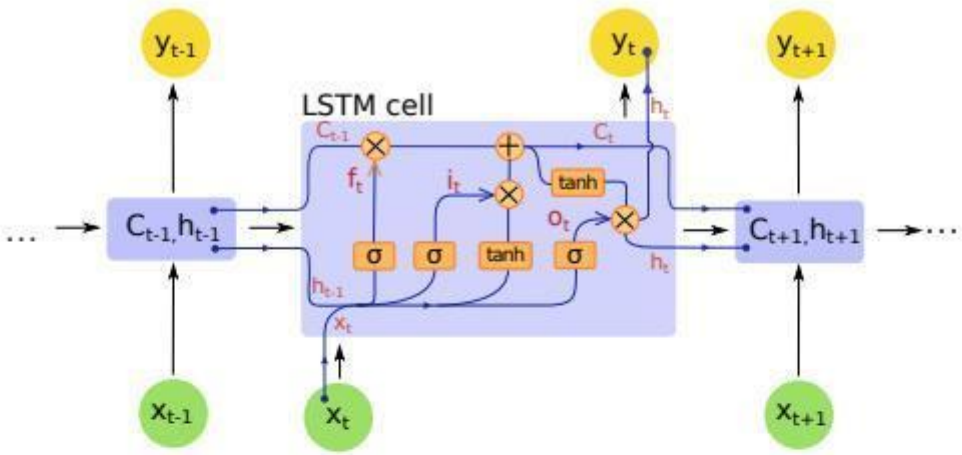


Figure 3. Long Short-Term Memory network and LSTM unit.

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= tanh(W_C[h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
y_t &= \sigma(W_y[h_{t-1}, x_t] + b_y) \\
h_t &= y_t * tanh(C_t)
\end{aligned}
$$

Такие модели считывают ввод данных последовательно. Если Вам интересна это тема, советую узнать также об алгоритме Seq2Seq. Нам нужна архитектура, в которой обработка последовательности производится сразу, что практически не оставляло бы места для потери информации. Да, такая архитектура реализована в кодировщике **модели Transformer**. Эта характеристика позволяет модели изучать контекст переменной на основе всего его окружения. Кроме того, по сравнению с рекуррентными нейросетями, чаще всего они быстрее.

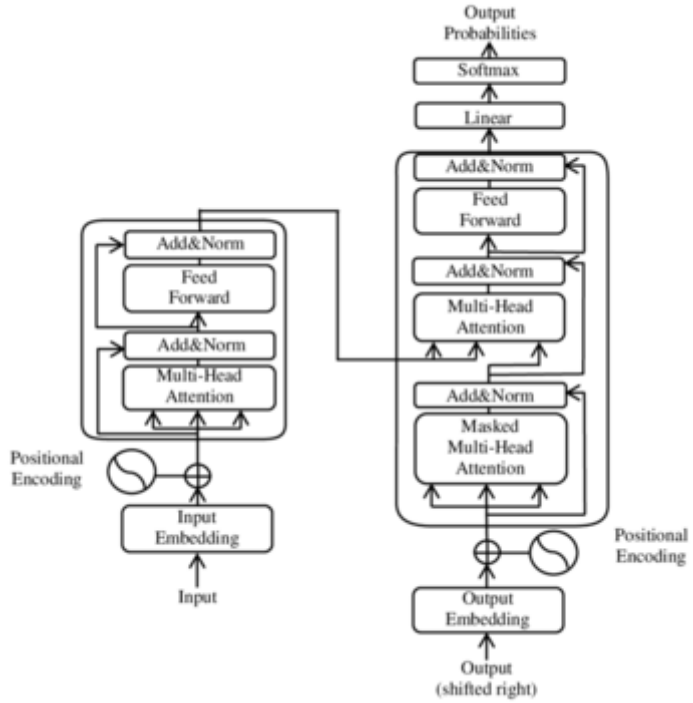
1.20 Алгоритм внимания и Transformer для временных рядов

Эта относительно новая идея слоя внимания очень интересна. Чаще всего такие модели применяются для обработки последовательностей текста.

В этой статье я опишу некоторое количество разновидностей моделей со слоем внимания, но углубляться в модели, используемые только для текста(BERT, GPT-2 и XLNet), конечно, я не буду. Хотя, с их недавним успехом в НЛП можно было бы ожидать широкой адаптации к прогнозированию временных рядов.

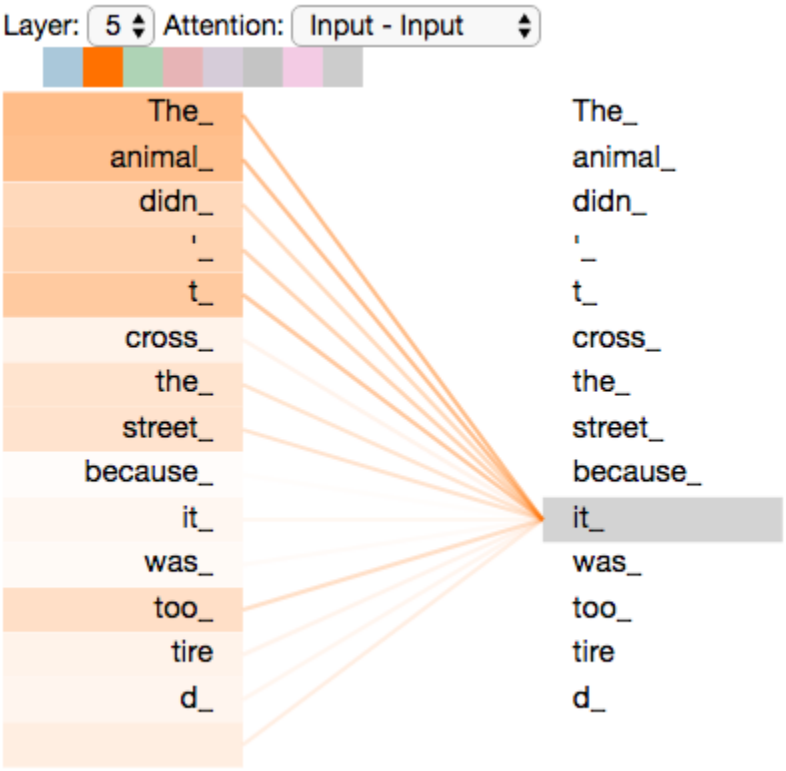
Оригинальная архитектура трансформера(Vanilla Transformer) является авто-энкодером. Кодер получает на вход последовательность с позиционной информацией. Декодер получает на вход часть этой последовательности и выход кодировщика. Но архитектура некоторых моделей на основе трансформера состоит только из encoder. Например, BERT.

Как можно заметить, модель состоит из преобразования входных векторов, позиционного кодирования, нормализации, слоёв прямого распространения, линейного слоя и слоёв внимания.



В психологии внимание — это концентрация сознания на одних стимулах, исключая другие. Слой внимания в нейросетях - математические

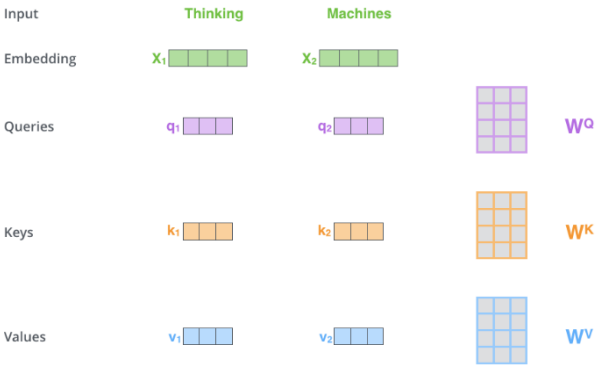
модели, позволяющие оценить взаимосвязь между

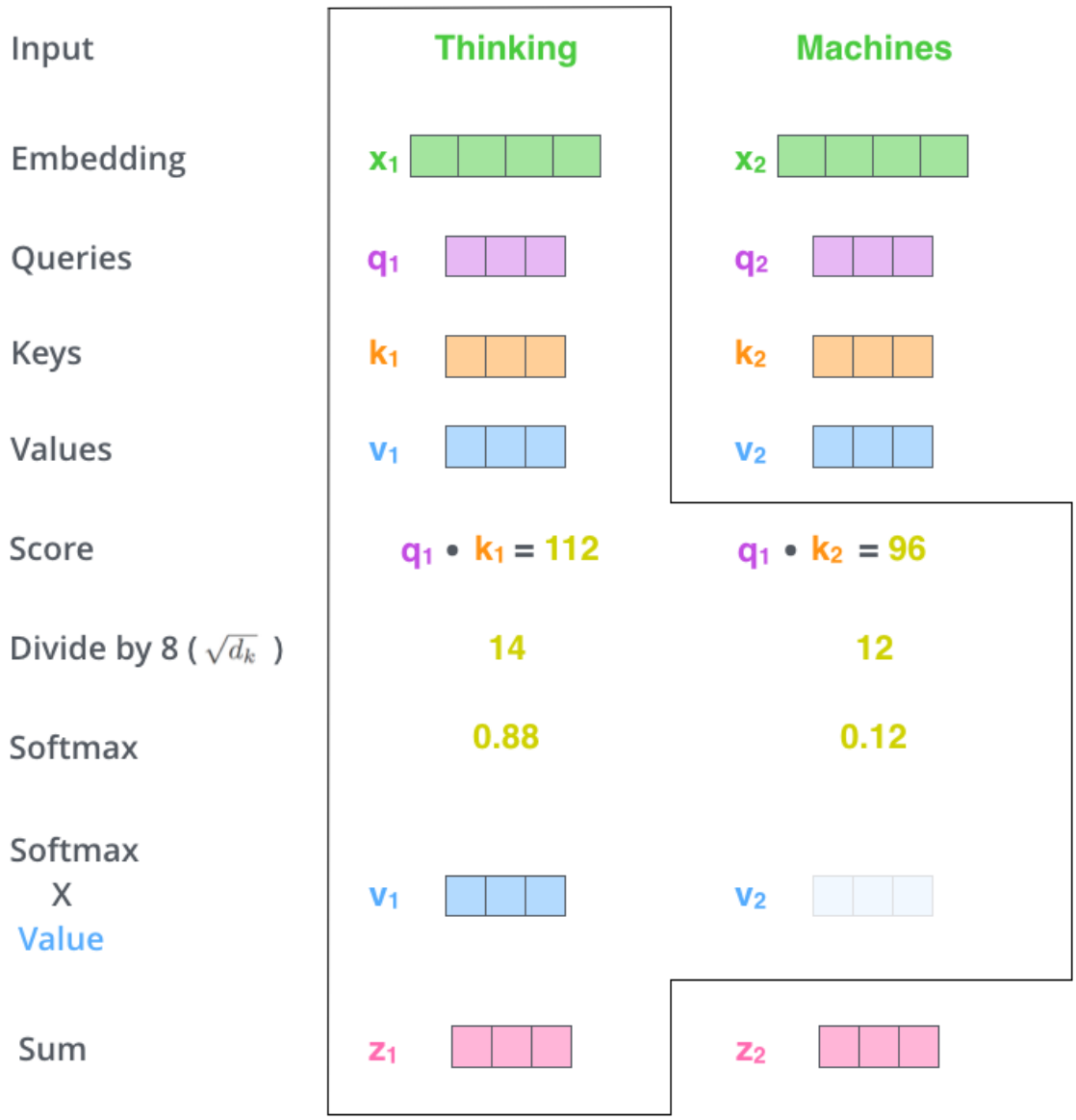


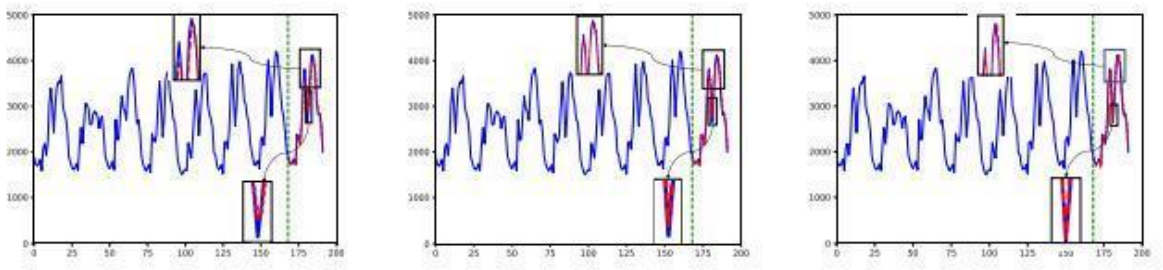
значениями.

Вот пример визуализации результата простого слоя внимания для последовательности слов. Таким образом, трансформеры сохраняют прямые связи со всеми предыдущими временными значениями, позволяя информации распространяться по гораздо более длинным последовательностям. Весь механизм помещён в формулу с функцией softmax

Более подробное описание слоя внимания







(a) Vanilla Transformer prediction (b) Sparse Transformer prediction (c) AST prediction

Виды трансформеров и их результаты в задаче прогнозирования временных рядов электричества

Существуют работы, посвященные моделям для прогнозирования временных рядов на основе трансформера. Давайте рассмотрим некоторые из них.

Первая работа, которая мне очень понравилась - Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case . Здесь на примере задачи прогнозирования гриппоподобных заболеваний автор сопоставил некоторые результаты различных моделей, работающих с временными рядами.

Архитектура очень похожа на оригинальный трансформер. В качестве оптимизатора в этой работе использовался Adam. Для регуляризации авторы добавили dropout и dropout rate - 0.2 для каждого слоя.

Посмотрим на результаты модели, оценка которой взята с RMSE и с коэффициентом корреляции

Table 1. Summary of model performances with relative change with respect to baseline model.

Model	Pearson Correlation	RMSE
ARIMA	0.769 (+0 %)	1.020 (-0 %)
LSTM	0.924 (+19.9 %)	0.807 (-20.9 %)
Seq2Seq+attn	0.920 (+19.5 %)	0.642 (-37.1 %)
Transformer	0.928 (+20.7 %)	0.588 (-42.4 %)

В следующей работе - Adversarial Sparse Transformer for Time Series Forecasting, нужно быть знакомым с генеративно-состязательными нейросетями, потому, что автор применил очень интересную идею создания генеративной модели для прогнозирования.

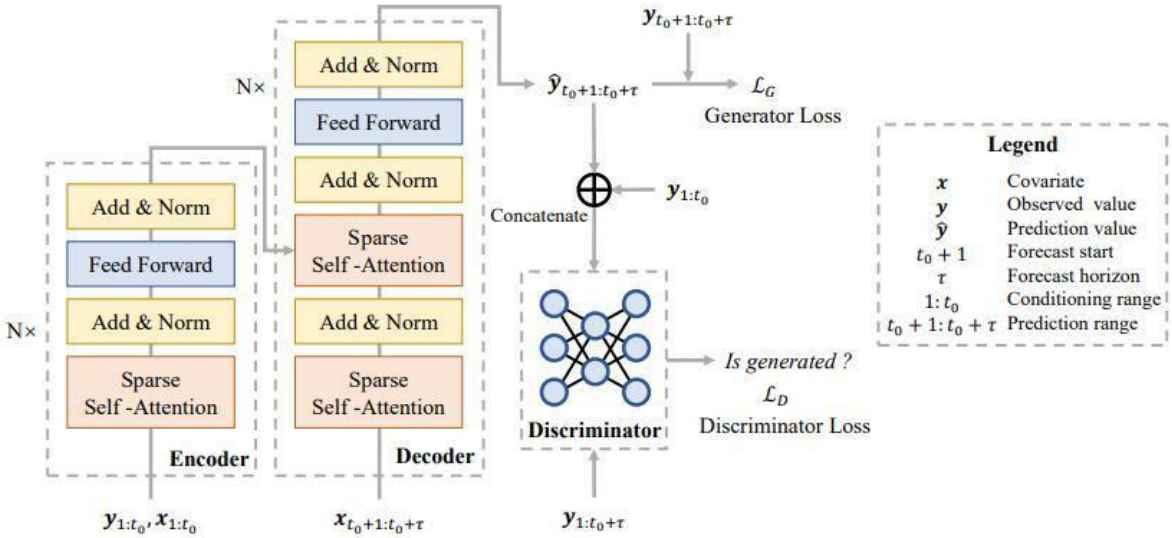


Figure 2: Architecture of the Adversarial Sparse Transformer Model

Применяется стохастический градиентный спуск

$$\arg\min_{\mathcal{G}}\max_{\mathcal{D}} \lambda\mathcal{L}_{adv}(\Theta^{\mathcal{G}},\Theta^{\mathcal{D}}) + \mathcal{L}_{\rho}(\Theta^{\mathcal{G}}),$$

$$\mathcal{L}_{adv}(\Theta^{\mathcal{G}},\Theta^{\mathcal{D}}) = \mathbb{E}[\log(\mathcal{D}(\mathbf{Y}_{real})] + \mathbb{E}[\log(1-\mathcal{D}(\mathbf{Y}_{fake}))],$$

$$\mathcal{L}_{\rho}(\Theta^{\mathcal{G}}) = 2\sum_{i=0}^{S}\sum_{t=t_0+1}^{t_0+\tau} P_{\rho}(y_{i,t},\hat{y}_{i,t}), \quad P_{\rho}(y_{i,t},\hat{y}_{i,t}) = \Delta y_{i,t}(\rho I_{\hat{y}_{i,t}>y_{i,t}} - (1-\rho)I_{\hat{y}_{i,t}\leq y_{i,t}}),$$

$$\Delta y_{i,t} = (\hat{y}_{i,t} - y_{i,t}), \mathbf{Y}_{fake} = (\mathbf{Y}_{1:t_0} \circ \hat{\mathbf{Y}}_{t_0+1:t_0+\tau}), \mathbf{Y}_{real} = (\mathbf{Y}_{1:t_0+\tau}),$$

Оптимизация

где: ΘG - параметры генератора , ΘD - параметры дискриминатора .

Обучилась нейросеть на данных часового ряда потребления электроэнергии (electricity).

В результате, ошибка обычного трансформера примерно в 3 раза меньше, чем ARIMA (0.036), а ошибка модели описанной выше в 4 раза меньше, чем ARIMA (0.025)

Вообще, большее количество типов данных сейчас предсказывается с использованием глубоких нейросетей. Некоторые архитектуры позволяют это сделать и для временных рядов.

Задание:

Подготовить выборку для прогнозирования временных рядов. Провести разведочный анализ данных. Сделать выводы о характере данных. Написать нейросеть для прогнозирования временных рядов. Оценить качество нейросети. Подготовить итоговый отчет.

Технология выполнения работы:

1. Выберите базу из открытых источников с временными рядами и проведите прогнозирование этого ряда на 10 шагов.
2. Спарсите, разделите и нормализуйте данные
3. Разделите данные на обучающую и проверочную выборки
4. Напишите сеть для прогнозирования временного ряда
5. Сделайте визуализацию результата
6. Напишите ваши результаты и выводы

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету::

- Подготовлен датасет
- Создана модель прогнозирования временных рядов
- Подготовлен отчет с выводами

Контрольные вопросы:

1. Что такое регрессионный анализ?
2. Чем отличается регрессия от классификации?
3. Какие виды зависимостей могут быть выявлены с использованием регрессионного анализа?
4. Что такое временной ряд и какие основные характеристики у него могут быть?
5. Какие методы прогнозирования временных рядов существуют, помимо нейронных сетей?
6. Что такое нейронные сети и как они могут быть применены для прогнозирования временных рядов?
7. Какие типы нейронных сетей чаще всего используются для прогнозирования временных рядов?
8. Каким образом можно подготовить данные временных рядов для использования в нейронных сетях?
9. Что такое функция потерь в контексте прогнозирования временных рядов, и как она выбирается?
10. Как оценивается качество прогнозов временных рядов, полученных с использованием нейронных сетей?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.

Практическое занятие № 22

Тема практического занятия: Генеративное моделирование

Цель практического занятия: изучение различных архитектур генеративных моделей, таких как вариационные автокодировщики (VAE), генеративные состязательные сети (GAN), авторегрессивные модели и другие.

В результате выполнения данной работы обучающийся должен
уметь:

- разрабатывать генеративно – состязательные модели;
- проводить оценку генеративно – состязательных моделей;
- обучать генеративно – состязательные нейросети и выполнять постпроцессинг данных, полученных на выходе из нейросети

знать:

- особенности и архитектуру автоэнкодеров и gan;
- понятия дискриминатор и генератор, их функции;
- метрики и функции потерь, используемые при обучении генеративных моделей

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Генеративно-состязательные сети, или сокращенно GAN, представляют собой подход к генеративному моделированию с использованием методов глубокого обучения, таких как сверточные нейронные сети.

Генеративное моделирование – это задача машинного обучения без учителя (неконтролируемое обучение), которая заключается в автоматическом обнаружении закономерностей и зависимостей во входных данных, которые можно было бы использовать для генерации на выходе новых примеров, которые могли бы быть непротиворечиво/правдоподобно присутствовать в оригинальном (исходном) наборе данных.

GAN представляют собой нетривиальный способ обучения генеративной модели, сводящий задачу к контролируемому процессу обучения (обучение с учителем), и состоящий из двух под-моделей: модели

генератора, которую мы обучаем генерировать новые примеры, и модели дискриминатора, которая пытается классифицировать примеры как реальные (из реального источника) или поддельные (сгенерированные). Две модели обучаются вместе в игре с нулевой суммой (антагонистической игре), состязательной, до тех пор, пока модель дискриминатора не начнет обманываться примерно в половине случаев, что означает, что модель генератора генерирует правдоподобные примеры.

GAN представляют собой захватывающую и быстро меняющуюся область, которая обещает создание генеративных моделей со способностью создавать реалистичные примеры для целого ряда проблемных областей, особенно в таких задачах обработки изображений, как перевод изображений из летних в зимние или изо дня в ночь, а также в создании реалистичных фотографий объектов, сцен и людей, которые люди не могут отличить от поддельных.

Генеративное моделирование представляет собой область в машинном обучении, которая фокусируется на создании моделей, способных генерировать новые данные, имитируя структуру и статистику набора обучающих данных. Эти модели применяются в различных задачах, таких как генерация изображений, текстов, музыки и других видов данных. Вот основные теоретические аспекты генеративного моделирования:

1) Вероятностные Генеративные Модели (Probabilistic Generative Models):
   Генеративные модели работают в вероятностном контексте, представляя собой вероятностные распределения, которые описывают структуру данных. Обучение таких моделей часто сводится к настройке их параметров так, чтобы они максимизировали правдоподобие (likelihood) обучающих данных.
2) Генеративные Состязательные Сети (GAN):
   GAN являются одной из ключевых архитектур в генеративном моделировании. Они включают в себя две нейронные сети - генератор и дискриминатор, которые соревнуются друг с другом. Генератор создает данные, а дискриминатор пытается различать настоящие данные от сгенерированных. Процесс обучения приводит к согласованности между ними.
3) Вариационные Автокодировщики (VAE):
   VAE являются еще одной популярной архитектурой генеративных моделей. Они основаны на комбинации автокодировщика и вероятностной модели. VAE обучаются так, чтобы создавать латентное пространство, из которого можно генерировать новые данные.
4) Марковские Случайные Поля (Markov Random Fields):
   Марковские случайные поля представляют собой графические модели, где вершины представляют элементы данных, а ребра - зависимости между ними. Они могут использоваться для моделирования структуры данных и восстановления пропущенных значений.
5) Функции Плотности Распределения (PDF):

Генеративные модели часто оцениваются по тому, насколько хорошо они могут приблизить функцию плотности распределения (PDF) данных. Чем ближе оценка к реальной PDF, тем лучше модель.

6) Функции Потерь (Loss Functions):
   Функции потерь определяют, насколько хорошо модель справляется с задачей генерации. Они могут включать в себя правдоподобие данных, дивергенцию Кульбака-Лейблера и другие метрики.
7) Этические Вопросы:
   Генеративное моделирование также вызывает этические вопросы, такие как создание фейковых данных, проблемы прозрачности в решениях моделей и вопросы приватности.

Генеративное моделирование является активным направлением исследований, и новые методы и архитектуры регулярно предлагаются для улучшения способности моделей создавать высококачественные и разнообразные данные.

Задание: Подготовить датасет с изображениями. Обработать данные, разделить на выборки. Написать нейросеть для генерации изображений котов. Архитектура нейросети – GAN. Обучить модель, сгенерировать изображения и прикрепить в отчет.

Технология выполнения работы:

1. Скачайте с Kaggle датасет для генерации котов
2. Разработайте архитектуру генеративно – состязательной модели
3. Напишите нейросеть и обучите на любом удобном фреймворке
4. Сгенерируйте 20 изображений котов
5. В отчете прикрепите изображения, архитектуру модели и код.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Подготовлен датасет;
- Создана модель генерации изображений;
- Подготовлен отчет с выводами

Контрольные вопросы:

1. Какие существуют виды нейросетей для генерации изображений?
2. Расскажите про GAN
3. Что определяют функции потерь

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.

Практическое занятие № 23

Тема практического занятия: Классификация, обнаружение и сегментация объектов

Цель практического занятия: изучение основных задач компьютерного зрения, методов и моделей, а также их практического применения в различных областях.

В результате выполнения данной работы обучающийся должен уметь:

- разрабатывать нейронные сети для решения задачи классификации;
- разрабатывать нейронные сети для решения задачи детекции
- разрабатывать нейронные сети для решения задачи сегментации
- оценивать модели глубокого обучения в области компьютерного

знать:

- особенности нейросетей для продвинутой работы с изображениями;
- архитектуры нейросетей для работы с объектами компьютерного зрения;
- метрики и функции потерь, используемые в компьютерном зрении

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Компьютерное зрение — это область искусственного интеллекта (ИИ), занимающаяся разработкой методов и технологий, позволяющих компьютерам анализировать, понимать и интерпретировать визуальную информацию, такую как изображения и видео.

1) Классификация объектов:

- *Определение:* Классификация в компьютерном зрении - это задача присвоения изображению или объекту на изображении одного или нескольких предопределенных классов.
- *Методы:* Используются методы машинного обучения, в частности, сверточные нейронные сети (CNN), для извлечения признаков и принятия решения о принадлежности объекта к определенному классу.

- *Применение:* Распознавание лиц, классификация объектов на медицинских изображениях, автоматическая категоризация контента.

2) Обнаружение объектов:
- *Определение:* Задача обнаружения объектов заключается в поиске и локализации объектов на изображении и присвоении им соответствующих классов.
- *Методы:* Обычно применяются сверточные нейронные сети, включая Region-based CNN (R-CNN), Faster R-CNN, и Single Shot MultiBox Detector (SSD), которые способны выполнять как обнаружение, так и классификацию объектов.
- *Применение:* Автомобильные системы помощи в вождении, системы видеонаблюдения, медицинское оборудование.

3) Сегментация объектов:
- *Определение:* Задача сегментации заключается в разделении изображения на сегменты (регионы), представляющие собой отдельные объекты, и присвоении каждому пикселю соответствующего класса.
- *Методы:* Методы сегментации могут быть основаны на сверточных нейронных сетях, таких как U-Net, а также на методах графовой резки и др.
- *Применение:* Медицинская сегментация (например, сегментация органов на изображениях МРТ), автоматическая обработка изображений в производственных процессах.

4) Сверточные нейронные сети (CNN):
- *Определение:* CNN являются ключевым инструментом для решения задач классификации, обнаружения и сегментации объектов в компьютерном зрении. Они характеризуются использованием сверток для извлечения пространственных признаков из входных данных.
- *Архитектура:* Включает слои свертки, слои подвыборки (pooling), полносвязные слои и слои активации.
- *Применение:* В различных задачах компьютерного зрения, начиная от классификации изображений и заканчивая сегментацией.

5) Метрики оценки производительности:
- *Точность (Accuracy):* Отношение правильно классифицированных объектов к общему количеству объектов.
- *Точность обнаружения (Detection Accuracy):* Доля правильно обнаруженных объектов среди всех действительно существующих.
- *IoU (Intersection over Union):* Мера, оценивающая перекрывание областей, используется для оценки качества сегментации.

6) Этические и социальные вопросы:
- *Конфиденциальность:* Использование технологий компьютерного зрения может вызывать вопросы о защите личной жизни.
- *Антибиас:* Возможность внедрения предвзятости в решения, основанные на данных.

- *Безопасность:* Риск злоупотребления технологией в недобросовестных целях.

Задание: Подготовить датасет для детекции летающих объектов. Построить одностадийную и двухстадийную модели для детекции летающих объектов. Провести сравнение работы моделей. Подготовить итоговый отчет с результатами проделанной работы.

Технология выполнения работы:

1. Подготовьте два датасета для детекции летающих объектов. Один в формате YOLO v3, другой в формате RetinaNet.
2. Напишите и обучите модель архитектуры Yolo
3. Напишите и обучите модель архитектуры RetinaNet
4. Проведите сравнение работы двух моделей
5. Составьте отчет с выводами о проведенных экспериментах.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Подготовлен датасет
- Созданы модели детекции
- Подготовлен отчет с выводами

Контрольные вопросы:

1. Какие существуют виды нейросетей для детекции изображений?
2. Какие существуют виды нейросетей для сегментации изображений?
3. Какие существуют виды нейросетей для классификации изображений?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation, 2018.

Практическое занятие № 24

Тема практического занятия: Применение нейронных сетей при кластеризации данных

Цель практического занятия: изучение особенностей применения глубоких нейронных сетей для кластеризации данных.

В результате выполнения данной работы обучающийся должен
уметь:

- разрабатывать модели кластеризации данных;
- оценивать качество работы моделей для кластеризации данных
- проводить предпроцессинг и постпроцессинг для решения задачи кластеризации

знать:

- особенности и архитектуру нейросетей для кластеризации;
- метрики оценки моделей кластеризации;
- способы улучшения качества работы моделей кластеризации

Общие теоретические сведения

Кластеризация (англ. cluster analysis) — задача группировки множества объектов на подмножества (кластеры) таким образом, чтобы объекты из одного кластера были более похожи друг на друга, чем на объекты из других кластеров по какому-либо критерию. Задача кластеризации относится к классу задач обучения без учителя.

Анализировать кластеры можно по следующим критериям:

- Центр класса.
- Среднее отклонение от центра.
- Гистограмма отклонения от центра. Дает представление о разбросе кластеров.
- Расстояние до других центров классов. Показывает, насколько текущий кластер обособлен от других.
- Гистограмма расстояния до точек других классов.
- Минимальная, средняя и максимальная близость чужих точек.

Основная метрика — это расстояние между элементами кластеров. Количество кластеров изначально задается самостоятельно, а потом определяется оптимальное количество.

Применение нейронных сетей при кластеризации данных предоставляет возможность эффективного выделения структур и закономерностей в больших объемах информации. Ниже приведены основные аспекты применения нейронных сетей в кластеризации данных:

1) Самоорганизующиеся карты (SOM):
    - SOM представляют собой нейронные сети, используемые для проекции многомерных данных на двумерное или трехмерное пространство.

- Эти карты позволяют выделить структуры и кластеры в данных, а также сохранить топологические отношения между объектами.

2) Автоассоциативные нейронные сети:
   - Автоассоциативные нейронные сети используются для реконструкции входных данных, что позволяет выявить внутренние закономерности и кластеры.
   - Автоассоциативные сети могут служить эффективными методами снижения размерности данных.
3) Глубокие нейронные сети (Autoencoders):
   - Автоэнкодеры представляют собой глубокие нейронные сети, которые могут выявлять сложные структуры в данных.
   - После обучения они могут использоваться для получения представлений данных в пространстве меньшей размерности, что помогает в кластеризации.
4) Кластеризация с использованием выходных слоев:
   - Нейронные сети могут быть настроены так, чтобы выходные слои предоставляли представления объектов, которые затем используются для кластеризации.
   - Кластеризация может быть выполнена классическими методами, такими как метод k-средних, с использованием этих представлений.
5) Оценка производительности:
   - Метрики оценки производительности (индекс силуэта, adjusted Rand index) используются для измерения качества кластеризации, полученной с использованием нейронных сетей.
   - Это важно для определения эффективности алгоритма и его применимости к конкретным данным.
6) Анализ результатов:
   - После кластеризации с использованием нейронных сетей проводится анализ полученных кластеров и их интерпретация в контексте задачи.
7) Этические аспекты:
   - Важно учитывать этические аспекты, связанные с использованием нейронных сетей при обработке данных, такие как конфиденциальность и безопасность информации.
8) Примеры применения:
   - Использование нейронных сетей для кластеризации в областях, таких как медицина, финансы, биотехнологии, розничная торговля и другие.

Применение нейронных сетей в кластеризации данных обеспечивает гибкий и мощный инструмент для анализа и структурирования информации, что позволяет автоматизировать процессы выделения групп похожих объектов.

Задание: Подготовить данные из набора mnist для решения залачи кластеризации. Построить модель кластеризации цифр из датасета mnist. Оценить качество работы модели. Подготовить итоговый отчет с результатами проделанной работы.

Технология выполнения работы:

1. Сделать классификацию цифр mnist, используя только Xtrain (без Ytrain).
2. Предобучите автокодировщик
3. Возьмите предобученный encoder
4. Кластеризовать скрытое пространство, полученное из encoder (т.е. Кластеризовать изображения, так как при подаче изображения на вход encoder'у получится скрытое пространство)
5. Протестируйте модель на Xtest и Ytest
6. Составьте отчет о проделанной работе

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Написана модель для кластеризации данных;
- Подготовлен отчет с выводами

Контрольные вопросы:

1. Какие существуют признаки для кластеризации?
2. Как организовать кластеризацию с помощью sklearn?
3. Как организовать кластеризацию нейросетью?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.

Практическое занятие № 25

Тема практического занятия: Интеграция модели машинного обучения в приложение

Цель практического занятия: изучение способов для осуществления интеграции моделей машинного обучения в приложения

В результате выполнения данной работы обучающийся должен

уметь:

- проводить интеграцию ml – моделей в приложение;
- подготавливать модели машинного обучения для запуска в production среде;
- ускорять работу нейросетей

знать:

- особенности интеграции ml – моделей в веб и графические приложения;
- понятия квантизации и прунинга;
- способы построения асинхронных api для работы с тяжеловесными моделями машинного обучения

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

Общие теоретические сведения

Выделяют следующие паттерны внедрения моделей машинного обучения в production:

- Модель как услуга или сервис (Model-as-Service);
- Модель как зависимость (Model-as-Dependency);
- Предварительный расчет (Precompute);
- Модель по запросу (Model-on-Demand);
- Гибридная модель обслуживания (Hybrid Model Serving) или Федеративное обучение (Federated Learning)

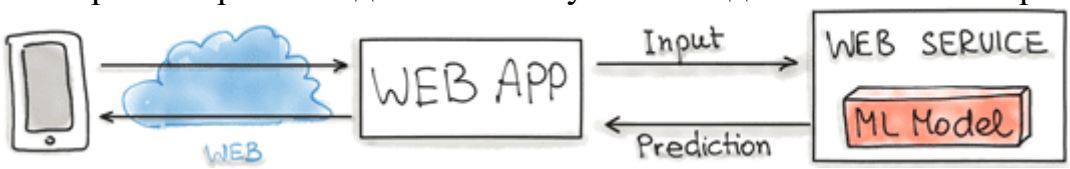
Критерий	**ML-модель**		
Обслуживание и версионирование (Service & Versioning)	Вместе с приложением-потребителем	Независимо от приложения-потребителя	
Доступность во время компиляции или выполнения (Compile/Runtime Availability)	Доступна во время сборки и выполнения (Build and runtime available)	Доступно удаленно через REST API или RPC-интерфейсы	Доступна во время выполнения

Шаблон обслуживания (Serving Pattern)	Модель как зависимость (Model-as-Dependency)	Модель как сервис (Model-as-Service)	Предварительный расчет (Precompute) и Модель по запросу (Model on Demand)

Гибридная модель обслуживания (Hybrid Model Serving) или Федеративное обучение (Federated Learning)

1) **Модель как услуга (Model-as-Service)**

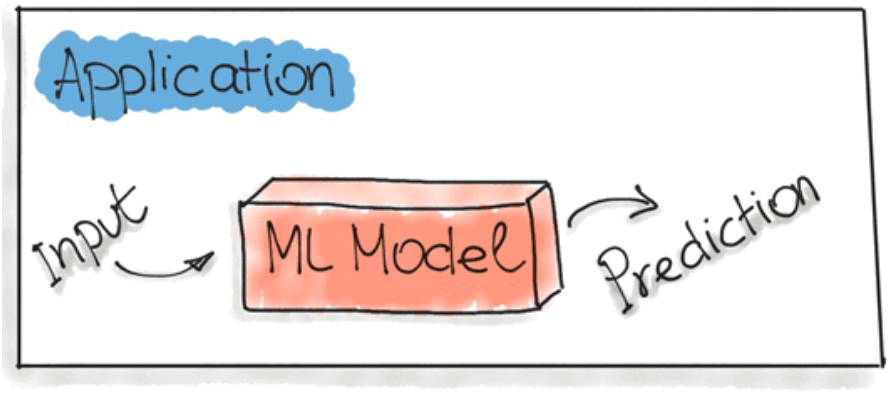
Это весьма распространенный шаблон для предоставления модели машинного обучения в качестве независимой услуги. Обычно это реализуется через заключение ML-модели и интерпретатора в выделенную веб-службу, к которой приложения-потребители данных запрашивают через REST API или удаленный вызов процедур (RPC, Remote Procedure Call). Такой паттерн пригоден для различных рабочих процессов машинного обучения, от пакетного прогнозирования до онлайн-обучения модели в потоковом режиме.



Машинное обучение как услуга — самый простой шаблон MLOps

2) **Модель как зависимость (Model-as-Dependency)**

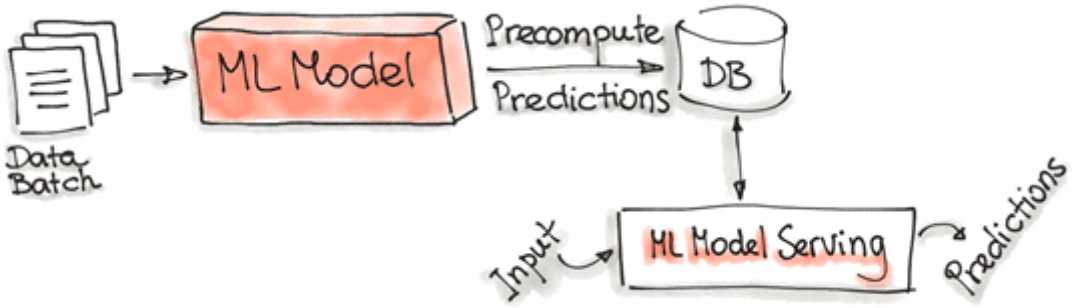
Это наиболее простой способ упаковать модель машинного обучения, которая рассматривается как зависимость внутри программного приложения. Например, приложение использует ML-модель как обычную зависимость от jar-файла, вызывая метод прогнозирования и передавая значения. Возвращаемое значение такого метода — некоторый прогноз, который выполняется предварительно обученной ML-моделью. Как правило, этот подход используется в задачах простого пакетного прогнозирования.



Модель как зависимость (Model-as-Dependency) — развертывание в production

3) **Предварительный расчет (Precompute)**

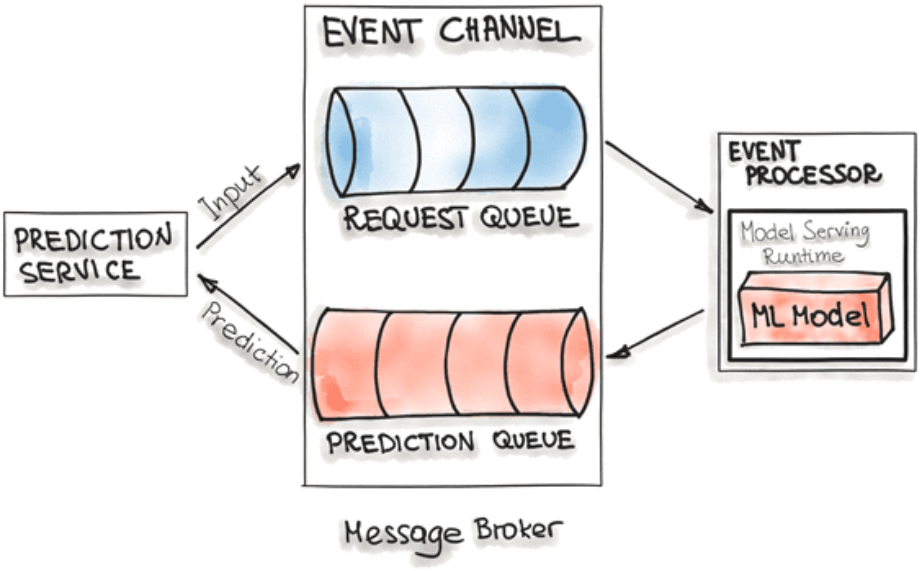
В этом случае прогнозы предварительно вычисляются с использованием уже обученной ML-модели для входящего пакета данных и сохраняются в базе. Далее к этой базе идет обращение при любом входном запросе, чтобы получить результат прогнозирования. С архитектурной точки зрения это похоже на Лямбда-шаблон, когда «горячая» обработка данных в потоковом режиме совмещается с «холодной», где пакеты исторических данных подгружаются из хранилища, в качестве которого обычно выступает Data Lake на Apache Hadoop. Подробнее о том, что такое лямбда-архитектура в Big Data системах, мы рассказывали здесь.



Лямбда-архитектура для ML-систем в MLOps

4) **Модель по запросу (Model-on-Demand) с Apache Kafka**

Этот вариант рассматривает модель машинного обучения как зависимость, доступную во время выполнения. Однако, в отличие от шаблона «Модель как зависимость», Model-on-Demand имеет собственный цикл выпуска и публикуется независимо. Для этой реализации обычно используется брокер сообщений, например, Apache Kafka или RabbitMQ (чем они отличаются, мы писали здесь). В этом случае применяется популярный в Big Data подход потоковой обработки событий (*event-stream processing*), когда данные представляются в виде потока событий, объединенные в канал. Каналы событий представляют собой очереди сообщений: входную и выходную. Брокер сообщений позволяет одному процессу записывать запросы на прогнозирование во входную очередь. Обработчик событий (*event processor*) содержит модель, обслуживающую среду выполнения, и ML-модель. Этот процесс подключается к брокеру, считывает из очереди запросы в пакетном режиме и отправляет их ML-модели для прогнозирования. Процесс обслуживания модели запускает генерацию прогнозов для входных данных и записывает полученные прогнозы в выходную очередь. Оттуда результаты прогнозирования отправляются в сервис, который инициировал запрос.

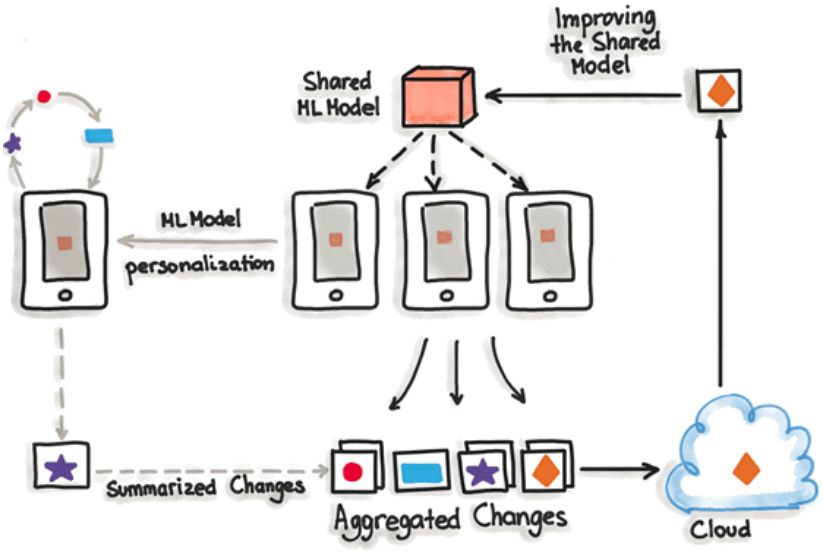


ML-система на Apache Kafka: практика MLOps

5) **Гибридная модель обслуживания (Hybrid Model Serving)**

Уникальность федеративного обучения или гибридного обслуживания в том, что оно работает со множеством ML-моделей, индивидуальных для каждого пользователя в дополнение к той, которая хранится на сервере. Серверная модель обучается только один раз с реальными данными и выступает в качестве начального образца для каждого пользователя. Далее она может видоизменяться в пользовательских вариациях, даже на мобильных устройствах, параметры которых сегодня позволяют обучать собственные ML-алгоритмы. Периодически пользовательские устройства отправляют на сервер уже обученные данные своей ML-модели, корректируя серверный вариант. Оттуда эти изменения могут быть распространены на остальных пользователей, с учетом предварительной проверки и тестирования на функциональность.

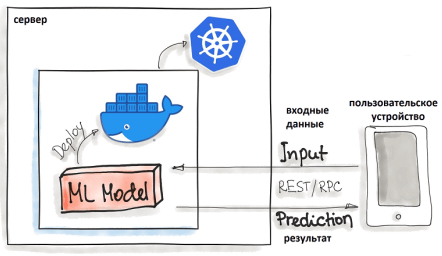
Главным плюсом такого гибридного подхода является то, что обучающие и тестовые датасеты, которые носят исключительно личный характер, никогда не покидают пользовательские устройства, сохраняя при этом все доступные данные. Это позволяет обучать высокоточные ML-модели без необходимости хранить гигабайты данных в облаке. Однако, стоит помнить, что обычные алгоритмы машинного обучения ориентированы на однородные и большие датасеты, которые обрабатываются на мощном оборудовании и всегда доступны для обучения. Современные мобильные устройства пока еще менее мощные, чем специализированные Big Data кластера, а также обучающие датасеты распределены по миллионам устройств, которые не всегда доступны. С учетом этого был создан отдельный фреймворк – TensorFlow Federated, облегченная форма TensorFlow для федеративного обучения.



Гибридная модель обслуживания (Hybrid Model Serving) или Федеративное обучение (Federated Learning) ML-моделей

С учетом вышерассмотренных паттернов эксплуатации ML-моделей, выделяют следующие стратегии их внедрения в production-системы:

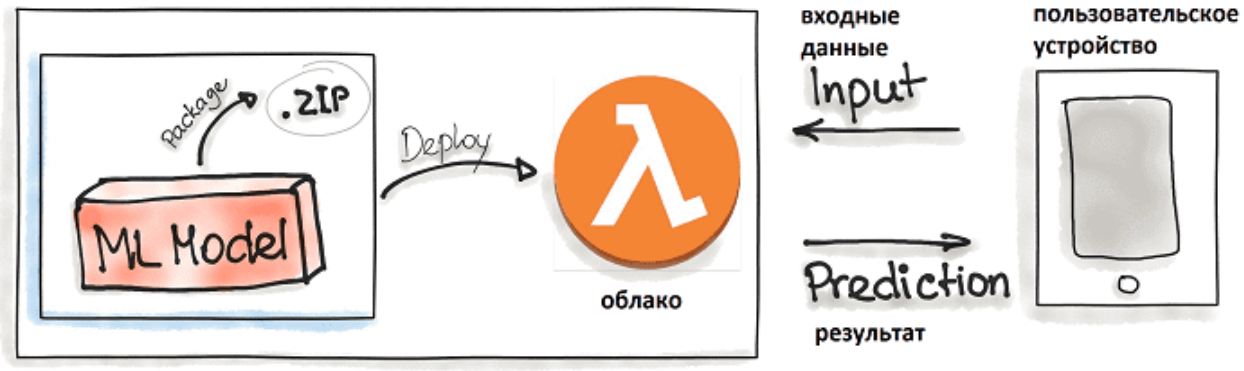
- *Развертывание с помощью Docker-контейнеров*, что подходит для легковесных идемпотентных ML-моделей без сохранения состояния. В этом случае код модели машинного обучения заключен в Docker-контейнер, который считается наиболее распространенной технологией контейнеризации для локального, облачного или гибридного развертывания. Оркестрация таких контейнеров обычно выполняется с помощью Kubernetes или альтернатив, таких как AWS Fargate. Функциональные возможности модели Machine Learning доступны через REST API, например, в виде приложения Flask. Таким образом, здесь активно используются наиболее популярные DevOps-технологии.



Развертывание ML-моделей с помощью Docker и Kubernetes

- *Бессерверные вычисления (serverless)*, когда код приложения и зависимости упаковываются в файлы .zip с одной функцией точки входа. Затем этой функцией могут управлять основные облачные провайдеры, такие как Azure Functions, AWS Lambda или Google Cloud Functions. Однако следует обратить внимание на возможные ограничения развертываемых артефактов, в частности, их размеры. Напомним, serverless-подход реализует PaaS- или FaaS- (функция как услуга, Function as a Service) стратегию, когда облако автоматически и динамически управляет выделением вычислительных ресурсов в зависимости от пользовательской нагрузки. При этом для выполнения

каждого запроса (вызова функции) создается отдельный контейнер или виртуальная машина, уничтожающиеся после выполнения. Преимуществом этого является избавление пользователей от работы по выделению и настройки серверов, в т.ч. виртуальных машин, контейнеров, баз данных, приложений, экземпляров сред выполнения. Все конфигурации и планирование вычислительных ресурсов для запуска кода по требованию или по событию скрыты от пользователей и управляются облаком. Бессерверный код может быть частью приложений, построенных на традиционной архитектуре, например, на микросервисах **[2]**. Обратной стороной этих достоинств являются зависимость от облачного провайдера, сложность в поиске причин случившихся ошибок из-за многослойной инкапсуляции внутреннего устройства всей системы, а также время на запуск облачной функции, которое может быть критично для бизнеса **[3]**.



Бессерверная стратегия внедрения ML-моделей в production

Задание

Интегрировать предварительно обученную модель машинного обучения в веб-приложение с использованием REST API.

Технология выполнения работы:

1. Выбор модели машинного обучения:
   - Выберите предварительно обученную модель, которую вы хотели бы интегрировать. Это может быть модель для задачи классификации, регрессии или другой задачи.
2. Разработка REST API:
   - Создайте простой веб-сервер с использованием фреймворка, такого как Flask или FastAPI.
   - Реализуйте REST API, предоставляющий точку входа для отправки данных и получения результатов работы модели.
3. Интеграция модели:
   - Настройте интеграцию выбранной модели в ваш веб-сервер. Обеспечьте возможность загрузки весов модели и выполнения предсказаний на основе входных данных.
4. Тестирование API:

- Напишите скрипты для тестирования вашего API. Протестируйте его с использованием различных входных данных, чтобы удостовериться в корректности работы.

5. Интеграция в веб-приложение:
   - Создайте простое веб-приложение (используйте HTML, CSS, JavaScript), которое взаимодействует с вашим REST API.
   - Реализуйте пользовательский интерфейс, позволяющий пользователям отправлять запросы к вашему API и получать результаты.
6. Обработка ошибок и безопасность:
   - Обеспечьте обработку ошибок в вашем веб-приложении и API.
   - Рассмотрите вопросы безопасности, такие как аутентификация и авторизация.
7. Документация:
   - Создайте краткую документацию по вашему API и веб-приложению, описывающую, как использовать их.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Создана/выбрана модель машинного обучения;
- Разработан REST API;
- Выполнена интеграция выбранной модели в веб-сервер

Контрольные вопросы:

1. Какие библиотеки и фреймворки широко используются для создания API для моделей машинного обучения?
2. Какие аспекты следует учитывать при выборе инструментов для интеграции моделей в приложение?
3. Какие этапы включает в себя процесс подготовки данных для интеграции моделей машинного обучения?
4. Что включает в себя процесс развертывания моделей в рабочей среде?
5. Почему мониторинг и обновление моделей важны после их интеграции в приложение?

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.

Практическое занятие № 26

Тема практического занятия: Использование образов и контейнеров Docker для развертывания моделей машинного обучения

Цель практического занятия: изучение принципов контейнеризации и изоляции среды для развертывания моделей машинного обучения

В результате выполнения данной работы обучающийся должен уметь:

- проводить интеграцию ml – моделей с помощью Docker;
- работать с Docker контейнерами и образами;
- работать с кластером Docker - Compose

знать:

- особенности интеграции ml – моделей с помощью Docker;
- принципы построения rest api;
- основные команды для работы с docker

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

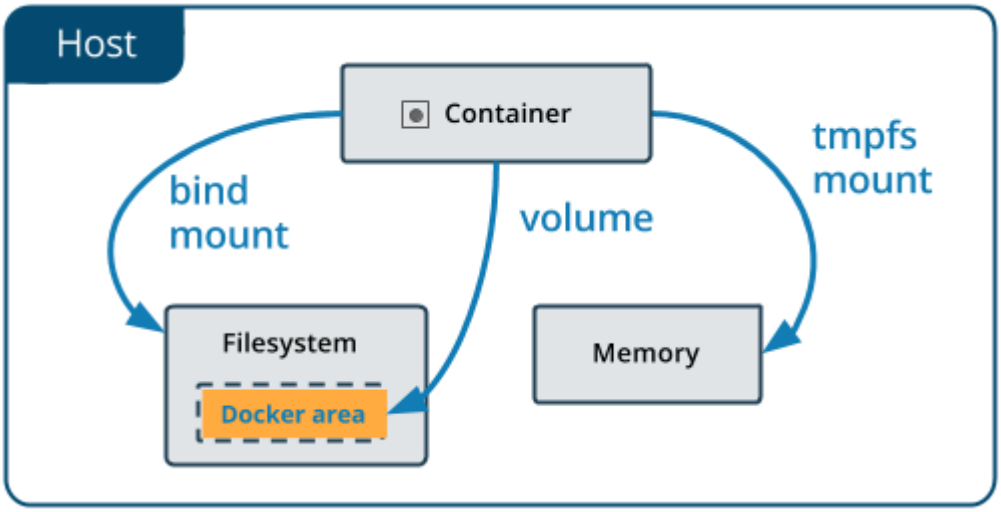
Общие теоретические сведения

В контейнерах Docker организовать работу с временными данными можно двумя способами. По умолчанию файлы, создаваемые приложением, работающим в контейнере, сохраняются в слое контейнера, поддерживающем запись. Для того чтобы этот механизм работал, ничего специально настраивать не нужно. Получается дёшево и сердито. Приложению достаточно просто сохранить данные и продолжить заниматься своими делами. Однако после того как контейнер перестанет существовать, исчезнут и данные, сохранённые таким вот нехитрым способом.

Для хранения временных файлов в Docker можно воспользоваться ещё одним решением, подходящим для тех случаев, когда требуется более высокий уровень производительности, в сравнении с тем, который достижим при использовании стандартного механизма временного хранения данных. Если вам не нужно, чтобы ваши данные хранились бы дольше, чем существует контейнер, вы можете подключить к контейнеру tmpfs — временное хранилище информации, которое использует оперативную память хоста. Это

позволит ускорить выполнение операций по записи и чтению данных. Часто бывает так, что данные нужно хранить и после того, как контейнер прекратит существовать. Для этого нам пригодятся механизмы постоянного хранения данных.

Существуют два способа, позволяющих сделать срок жизни данных большим срока жизни контейнера. Один из способов заключается в использовании технологии bind mount. При таком подходе к контейнеру можно примонтировать, например, реально существующую папку. Работать с данными, хранящимися в такой папке, смогут и процессы, находящиеся за пределами Docker. Вот как выглядят монтирование tmpfs и технология bind mount.



Минусы использования технологии bind mount заключаются в том, что её использование усложняет резервное копирование данных, миграцию данных, совместное использование данных несколькими контейнерами. Гораздо лучше для постоянного хранения данных использовать тома Docker.

Том — это файловая система, которая расположена на хост-машине за пределами контейнеров. Созданием и управлением томами занимается Docker. Вот основные свойства томов Docker:

- Они представляют собой средства для постоянного хранения информации.
- Они самостоятельны и отделены от контейнеров.
- Ими могут совместно пользоваться разные контейнеры.
- Они позволяют организовать эффективное чтение и запись данных.
- Тома можно размещать на ресурсах удалённого облачного провайдера.
- Их можно шифровать.
- Им можно давать имена.
- Контейнер может организовать заблаговременное наполнение тома данными.
- Они удобны для тестирования.

Тома можно создавать средствами Docker или с помощью запросов к API. Вот инструкция в Dockerfile, которая позволяет создать том при запуске контейнера.

VOLUME /my_volume

При использовании подобной инструкции Docker, после создания контейнера, создаст том, содержащий данные, которые уже имеются в указанном месте. Обратите внимание на то, что если вы создаёте том с использованием Dockerfile, это не освобождает вас от необходимости указать точку монтирования тома.
Создавать тома в Dockerfile можно и используя формат JSON.
Кроме того, тома можно создавать средствами командной строки во время работы контейнера.

1.21 Работа с томами из командной строки

1. Создать самостоятельный том можно следующей командой:
docker volume create —-name my_volume

2. Для того чтобы просмотреть список томов Docker, воспользуйтесь следующей командой:

docker volume ls

3. Исследовать конкретный том можно так:

docker volume inspect my_volume

4. Удалить том можно так:

docker volume rm my_volume

5. Для того чтобы удалить все тома, которые не используются контейнерами, можно прибегнуть к такой команде:

docker volume prune

6. Перед удалением томов Docker запросит у вас подтверждение выполнения этой операции. Если том связан с каким-либо контейнером, такой том нельзя удалить до тех пор, пока не удалён соответствующий контейнер. При этом, даже если контейнер удалён, Docker не всегда это понимает. Если это случилось — можете воспользоваться следующей командой:

docker system prune

Она предназначена для очистки ресурсов Docker. После выполнения этой команды у вас должна появиться возможность удалить тома, статус которых до этого определялся неправильно.

Для работы с томами вам, при вызове команды docker, часто придётся пользоваться флагами.

7. Например, для того чтобы создать том во время создания контейнера можно воспользоваться такой конструкцией:

docker container run --mount source=my_volume,
target=/container/path/for/volume my_image

В давние времена (до 2017 года) популярен был флаг --volume. Изначально этот флаг (ещё им можно пользоваться в сокращённом виде, тогда он выглядит как -v) использовался для самостоятельных контейнеров, а флаг --mount — в среде Docker Swarm. Однако, начиная с Docker 17.06, флаг --mount можно использовать в любых сценариях.

Надо отметить, что при использовании флага --mount увеличивается объём дополнительных данных, которые приходится указывать в команде, но, по нескольким причинам, лучше использовать именно этот флаг, а не --volume. Флаг --mount — это единственный механизм, который позволяет работать с сервисами или указывать параметры драйвера тома. Кроме того, работать с этим флагом проще.

В существующих примерах команд, направленных на работу с данными в Docker, вы можете встретить множество примеров употребления флага -v. Пытаясь адаптировать эти команды для себя, учитывайте то, что флаги --mount и --volume используют различные форматы параметров. То есть, нельзя просто заменить -v на --mount и получить рабочую команду.

Главное различие между --mount и --volume заключается в том, что при использовании флага --volume все параметры собирают вместе, в одном поле, а при использовании --mount параметры разделяются.

При работе с --mount параметры представлены как пары вида ключ-значение, а именно, это выглядит как key=value. Эти пары разделяют запятыми. Вот часто используемые параметры --mount:

type — тип монтирования. Значением для соответствующего ключа могут выступать bind, volume или tmpfs. Мы тут говорим о томах, то есть — нас интересует значение volume.

source — источник монтирования. Для именованных томов это — имя тома. Для неименованных томов этот ключ не указывают. Он может быть сокращён до src.

destination — путь, к которому файл или папка монтируется в контейнере. Этот ключ может быть сокращён до dst или target.

readonly — монтирует том, который предназначен только для чтения. Использовать этот ключ необязательно, значение ему не назначают.

Вот пример использования --mount с множеством параметров:

docker run --mount type=volume,source=volume_name,destination=/path/in/container,readonly my_image

Вот полезные команды, которыми можно пользоваться при работе с томами Docker:

- docker volume create
- docker volume ls
- docker volume inspect
- docker volume rm
- docker volume prune

Вот список часто используемых параметров для --mount, применимых в команде вида docker run --mount my_options my_image:

- type=volume
- source=volume_name
- destination=/path/in/container
- readonly

Задание:

Написать одностраничное веб – приложение с использованием ml – модели. Выполнить отладку веб – приложения. Обернуть приложение в Docker. Проверить работоспособность прилодения на сервере. Подготовить итоговый отчет о проделпнной работе.

Технология выполнения работы:

1. Возьмите модель и приложение из практического занятия 25.
2. Создайте docker-compose.yml файл для развертывания приложения с помощью docker-compose. Проверьте работоспособность кластера docker-compose.
3. Добавьте в приложение базу данных куда будут логгироваться запросы пользователя и ответы модели. Докеризуйте базу данных.

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Приложение обернуто в Docker;
- База данных обернута в Docker;
- Кластер docker-compose работает корректно

Контрольные вопросы:

1. Что такое docker?
2. Как развернуть приложение с помощью Docker Compose.
3. Как проверить работоспособность Кластер docker-compose

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.

Практическое занятие № 27

Тема практического занятия: Развёртывание модели машинного обучения в виде REST API

Цель практического занятия: научиться разворачивать модель машинного обучения в виде REST API

В результате выполнения данной работы обучающийся должен уметь:

- разрабатывать api в стиле rest;
- использовать rest api для построения веб – приложений и микросервисов;
- развертывать программные интерфейсы приложения в стиле rest на веб-сервере

знать:

- особенности и принципы rest – api;
- аналоги rest – api и принципы их работы;
- особенности клиент – серверной архитектуры;
- понятие протокола http и его разновидности

Перечень оборудования, необходимого для выполнения задания:

- Автоматизированные рабочие места по количеству обучающихся (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Автоматизированное рабочее место преподавателя (процессор Intel Core i7 или аналогичный, БП 700 Вт, 16 Гб ОЗУ, SSD 256 Гб, HDD 1 ТБ SATA 7200 rpm);
- Монитор 27” IPS, 1920x1080, 178°/178°, 250cd/m2, 1000:1, 5 ms, HDMI, DP (2 монитора на одно рабочее место)
- Интерактивная панель 86” с OPS ПК
- Программное обеспечение общего и профессионального назначения

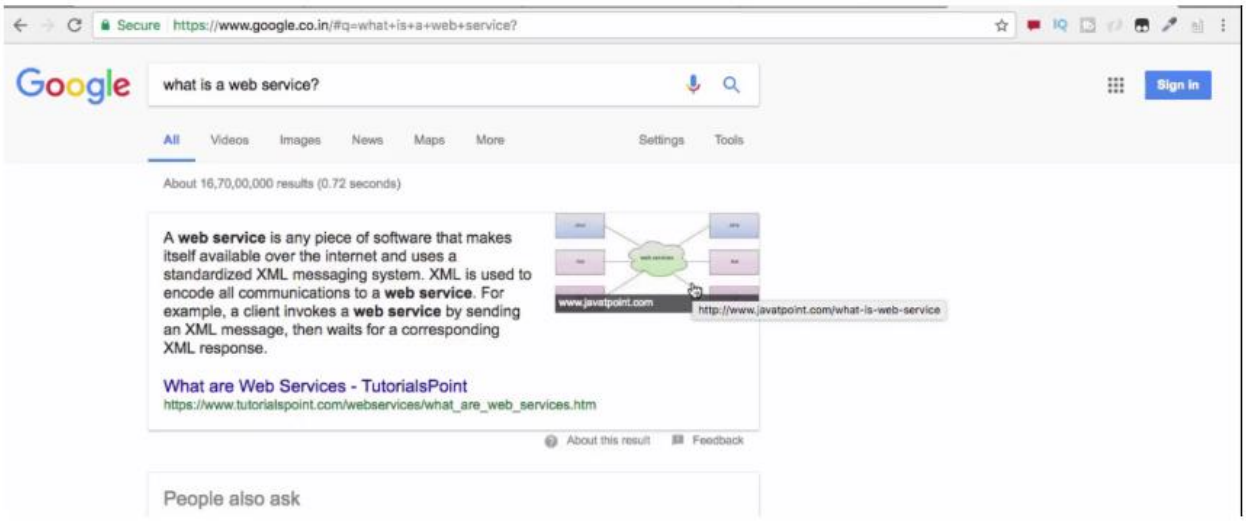
Общие теоретические сведения

REST означает REpresentational State Transfer (Википедия: «передача состояния представления»). Это популярный архитектурный подход для создания API в современном мире.

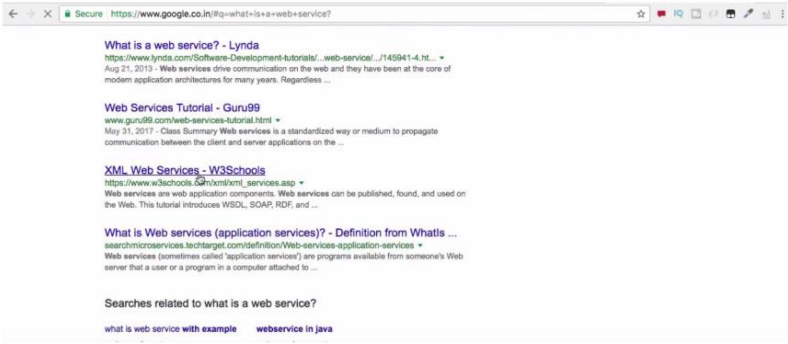
REST расшифровывается как REpresentational State Transfer. Это был термин, первоначально введен Роем Филдингом (Roy Fielding), который также был одним из создателей протокола HTTP. Отличительной особенностью сервисов REST является то, что они позволяют наилучшим образом использовать протокол HTTP. Теперь давайте кратко рассмотрим HTTP.

Краткий обзор HTTP

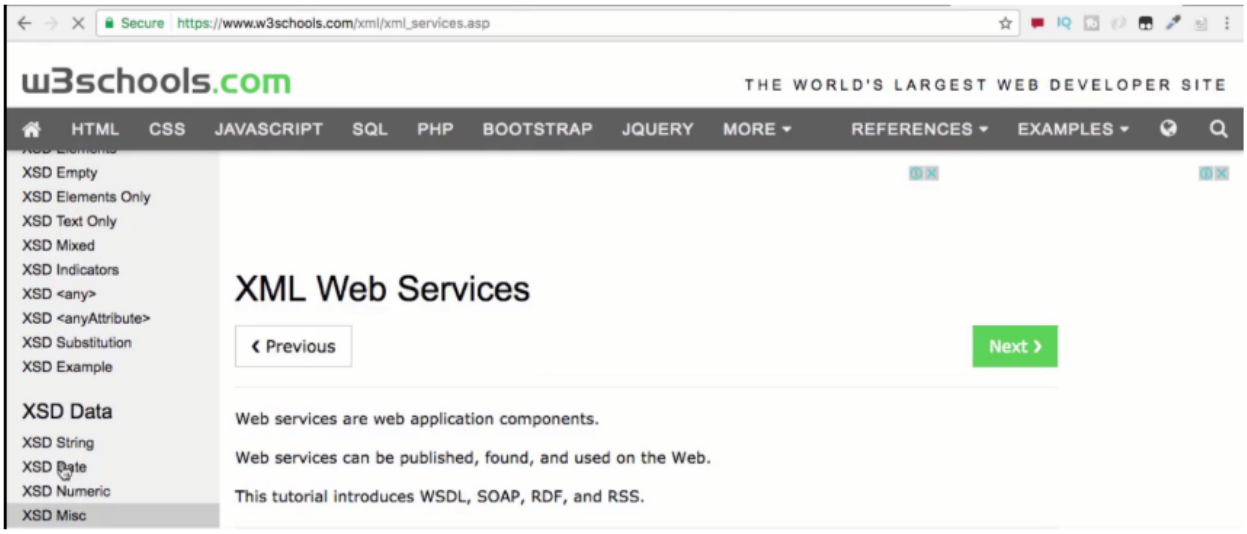
Давайте сначала откроем браузер и зайдем на веб-страницу:



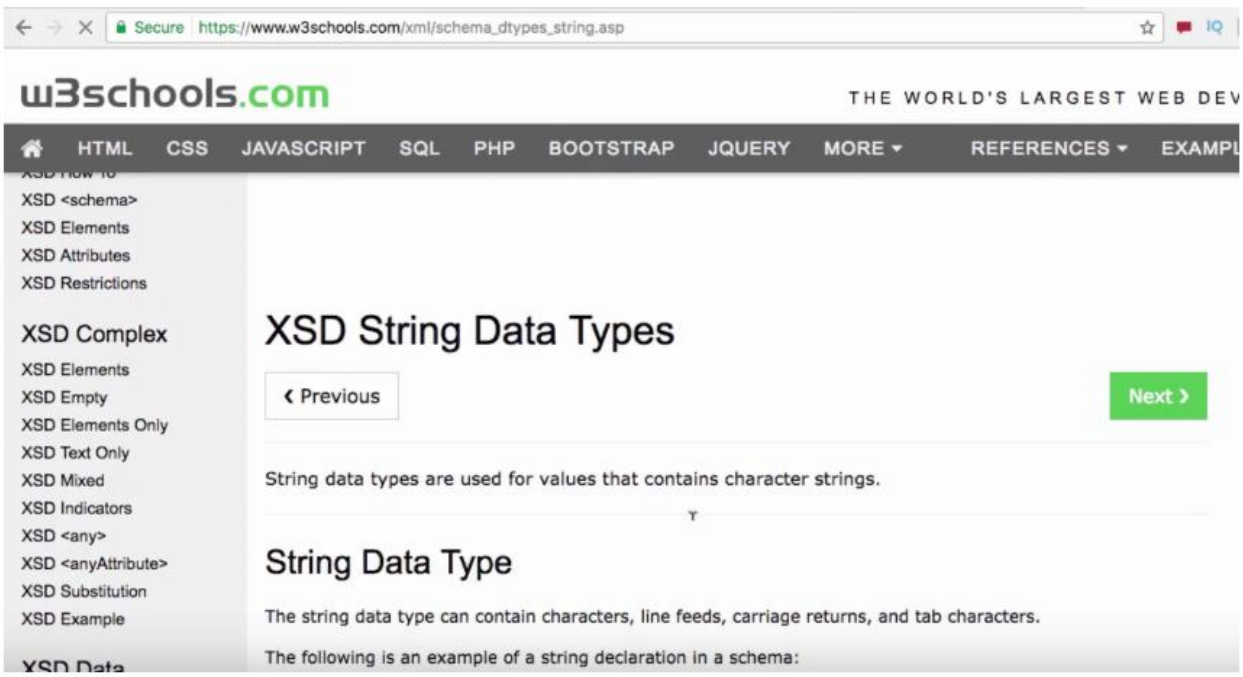
А затем щелкните на одной из страниц результатов:



Далее мы можем нажать на ссылку на странице, на которой мы оказались:



И перейти на другую страницу:



Когда мы просматриваем страницы в Интернете, за кулисами происходит много вещей. Ниже приведено упрощенное представление о том, что происходит между браузером и серверами, работающими на посещаемых веб-сайтах:

Протокол HTTP

Когда вы вводите в браузере URL-адрес, например www.google.com, на сервер отправляется запрос на веб-сайт, идентифицированный URL-адресом. Затем этот сервер формирует и выдает ответ. Важным является формат этих запросов и ответов. Эти форматы определяются протоколом **HTTP — Hyper Text Transfer Protocol**.

Когда вы набираете URL в браузере, он отправляет запрос GET на указанный сервер. Затем сервер отвечает HTTP-ответом, который содержит данные в формате **HTML — Hyper Text Markup Language**. Затем браузер получает этот HTML-код и отображает его на экране.

Допустим, вы заполняете форму, присутствующую на веб-странице, со списком элементов. В таком случае, когда вы нажимаете кнопку «**Submit**» (Отправить), HTTP-запрос **POST** отправляется на сервер.

HTTP обеспечивает базовый уровень для создания веб-сервисов. Поэтому важно понимать HTTP. Вот несколько ключевых абстракций.

Ресурс

Ресурс — это ключевая абстракция, на которой концентрируется протокол HTTP. Ресурс — это все, что вы хотите показать внешнему миру через ваше приложение. Например, если мы пишем приложение для управления задачами, экземпляры ресурсов будут следующие:

- Конкретный пользователь
- Конкретная задача
- Список задач

URI ресурса

Когда вы разрабатываете RESTful сервисы, вы должны сосредоточить свое внимание на ресурсах приложения. Способ, которым мы идентифицируем ресурс для предоставления, состоит в том, чтобы назначить ему URI — универсальный идентификатор ресурса. Например,

- Создать пользователя: **POST /users**
- Удалить пользователя: DELETE /users/1
- Получить всех пользователей: GET /users
- Получить одного пользователя: GET /users/1
- REST и Ресурсы

Важно отметить, что с REST вам нужно думать о приложении с точки зрения ресурсов. Определите, какие ресурсы вы хотите открыть для внешнего мира Используйте глаголы, уже определенные протоколом HTTP, для выполнения операций с этими ресурсами.

Вот как обычно реализуется служба REST:

- Формат обмена данными: здесь нет никаких ограничений. JSON — очень популярный формат, хотя можно использовать и другие, такие как XML;
- Транспорт: всегда HTTP. REST полностью построен на основе HTTP;
- Определение сервиса: не существует стандарта для этого, а REST является гибким. Это может быть недостатком в некоторых сценариях, поскольку потребляющему приложению может быть необходимо понимать форматы запросов и ответов. Однако широко используются такие языки определения веб-приложений, как WADL (Web Application Definition Language) и Swagger.

REST фокусируется на ресурсах и на том, насколько эффективно вы выполняете операции с ними, используя HTTP.

HTTP определяет следующую структуру запроса:

- строка запроса (request line) — определяет тип сообщения
- заголовки запроса (header fields) — характеризуют тело сообщения, параметры передачи и прочие сведения
- тело сообщения (**body**) — необязательное

HTTP определяет следующую структуру ответного сообщения (response):

- строка состояния (status line), включающая код состояния и сообщение о причине

- поля заголовка ответа (header fields)
- дополнительное тело сообщения (body)

Методы HTTP-запроса

Метод, используемый в HTTP-запросе, указывает, какое действие вы хотите выполнить с этим запросом. Важные примеры:

- GET: получить подробную информацию о ресурсе
- POST: создать новый ресурс
- PUT: обновить существующий ресурс
- DELETE: Удалить ресурс

Код статуса ответа HTTP

Код состояния всегда присутствует в ответе HTTP. Типичные примеры:

- 200 — успех
- 404 — страница не найдена

Задание: Подготовить генеративную модель для русского языка. Написать api endpoint для модели выбранной машинного обучения. Выполнить докеризацию API. Подготовить документацию к API. Подготовьте итоговый отчет о проделанной работе.

Технология выполнения работы:

1. Выберете любую генеративную языковую модель для русского языка и напишить api endpoint в стиле rest который принимает текст пользователя и возвращает ответ.
2. Докеризуйте api
3. Выполните развертывание программного интерфейса на сервере
4. Проверьте работоспособность
5. Добавьте документацию в Swagger для вашего эндпоинта

Указания по технике безопасности:

В процессе выполнения работы необходимо соблюдать правила безопасности и поведения на рабочем месте.

Требования к отчету:

- Написан API в стиле Rest
- База данных обернута в Docker

Контрольные вопросы:

1. Что Rest API?
2. Принципы Rest API
3. Какой протокол использует Rest API?
4. Особенности клиент – серверной архитектуры

Основные и дополнительные источники, электронные ресурсы:

1. Радченко, И.А, Николаев, И.Н. Технологии и инфраструктура Big Data. – СПб: Университет ИТМО, 2018.
2. Сенько, А. Работа с BigData в облаках. Обработка и хранение данных с примерами из Microsoft Azure. — СПб.: Питер, 2019. — 448 с.: ил.
3. Тесленко, И. Б., Губернаторов, А. М., Дигилина, О. Б., Крылов, В. Е. Большие данные. 2021.
4. The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation, 2018.