Работа с семиструктурированными данными

МОДУЛЬ 12

ШКОЛА БОЛЬШИХ ДАННЫХ

Классификация данных

- структурированные
- неструктурированные
  - ✓ семиструктурированные (**JSON**, **XML**, **текстовые**)
  - ✓ без структуры (**текст**, **двоичные**)

Семиструктурированные данные

- данные состоят из элементов
- элемент данных имеет структуру
- структура меняется от элемента к элементу
  - ✓ части структуры могут отсутствовать
- элемент данных имеет сложную структуру
  - ✓ вложенность (**структура в структуре**)
  - ✓ повторяющиеся элементы (**массивы**)

Как выглядит на практике

- **файлы** (форматов **JSON**, **XML**)
- результаты обращения к веб сервисам (как правило JSON)
- данные в NoSQL хранилищах (например, **Elasticsearch**)
- как правило, размер элемента данных не велик (**единицы килобайт**)

Проблематика

- представление для обработки
  - ✓ как представить вариативные по структуре данные
- представление для долгосрочного хранения
  - ✓ как **сохранить результаты** обработки
  - ✓ как **обеспечить эффективный доступ** к данным

Что дает Spark

- представление для обработки (**“dataframe”**)
  - ✓ поддержка **StructType**
  - ✓ поддержка **ArrayType**
- **парсинг** JSON -> dataframe
  - ✓ из **файла** или **RDD** (автоматическое построение или явная схема)
  - ✓ из поля **dataframe** (явная схема)
- "**продукты третьих фирм**" XML -> dataframe
- механизм **UDF**

Особенности работы с JSON

- удобно (автоматический **парсинг = построение схемы**)
- производительность достаточно высока
- нет возможности автоматической конвертации строк в числа
  - ✓ в отличие от CSV
- ошибки в схеме "не прощаются"
- схема строится "по наихудшему" варианту
  - ✓ поле имеет тип "**структура**"
    - ➢ если попадается "null" - получим тип "строка"
  - ✓ поле имеет тип "**массив**"
    - ➢ можем получить "не массив" (массив структур -> структура)
    - ➢ можем получить строку
- **как бороться**
  - ✓ отдельно обрабатывать записи с пустыми полями (**массив и структура**)
  - ✓ принудительно **конвертировать в строку**

Как работать с XML

- преобразуем XML -> JSON (с использованием **UDF**)
  - ✓ множество XML парсеров (например, **ElementTree**)
- далее работаем как с JSON
- **недостатки**
  - ✓ теряем возможность получения исходного XML (например, “namespaces”)
  - ✓ производительность хуже (за счет UDF)
- проверена на практике (на достаточно больших объемах)

Долгосрочное хранение семиструктурированных данных

- готового "хорошего" решения не существует
  - ✓ см. особенности выше
  - ✓ возможно, стоит проработать вариант с **ElasticSearch**

Компромиссные варианты

- "двоичный вид" (**key-value**)
- колончатое хранение (**wide column store**)
- композитный подход
  - ✓ таблица для наиболее часто используемых элементов
  - ✓ "двоичное" хранение элемента данных в поле таблицы

Хранение в "key-value"

- основной рабочий **вариант**
  - ✓ придумать ключ
  - ✓ данные - "значение" (в виде строки)
- **достоинства**
  - ✓ просто
  - ✓ эффективно в плане хранения
- **недостатки**
  - ✓ полное сканирование при поиске информации
  - ✓ полное отсутствие информации о структуре данных
  - ✓ необходимость парсинга данных перед обработкой

Хранение в колончатых базах

- “wide column store”
  - ✓ нет требования на "одинаковость" структуры записей в одной "таблице"
  - ✓ эффективное хранение "**пустоты**"
- как храним
  - ✓ каждый элемент данных - "**колонка**"
  - ✓ имя колонки - "путь" к элементу данных
- **достоинства**
  - ✓ **структура** каждой записи **сохранена**
  - ✓ возможен **доступ к любому полю** записи
- **недостатки**
  - ✓ **поиск** по-прежнему **неэффективен**
  - ✓ нужно что-то придумать для именования массивов
  - ✓ хранение имени колонки может быть "дороже" хранения значения

Композитное хранение

- используем **реляционную модель** (например, **Hive**)
- как храним
  - ✓ выделяем поля данных, по которым производится поиск
    - ➢ храним их в виде полей реляционной таблицы
  - ✓ всю запись дополнительно храним в одном поле (как строку)
- **достоинства**
  - ✓ не теряем информации
  - ✓ **эффективный поиск** по поисковым полям
- **недостатки**
  - ✓ долгая операция добавления поискового поля
  - ✓ **необходимость парсинга** данных перед обработкой
  - ✓ **дублирование** данных

Вопросы?