BIGDATA TEAM

HW #9: Cassandra + Spark

автор задания:
- Klemenkov Pavel, klemenkov@bigdatateam.org
- Big Data Instructor @ BigData Team
- Chief Data Scientist @ Nvidia, Data Platform

редакторы задания[1]:
- Николай Попов*, Ксения Пеньевская**, Александр Ким
- Big Data Mentor @ BigData Team
- *Data Engineer @ inDriver
- **Big Data Analyst

[1] Хочешь стать ментором и оставить след в истории Big Data? Тогда хорошо учись, помогай другим и дай нам знать о своем желании. Смело пиши преподавателям и менеджерам учебных курсов.

1. Описание задания

В этом задании вы будете разрабатывать схему данных в Cassandra для видео-сервиса, основанного на данных Movielens. От вас потребуется:

1. спроектировать и реализовать в Cassandra схему данных для получения ответов на поставленные вопросы;
2. написать запросы для получения ответов;
3. написать на Spark SQL код загрузки данных из HDFS в Cassandra.

Дополнительная информация

Запуск cqlsh на brain-client:

```
$ cqlsh brain-node1
```

Запуск Spark со Spark Cassandra Connector:

```
$ <options to run pyspark or spark-submit> --conf
spark.cassandra.connection.host=brain-node1 --packages
com.datastax.spark:spark-cassandra-connector_2.11:2.4.2 --conf
spark.cassandra.auth.username=<username>
```

Запуск CQL скриптов в автоматическом режиме будет производиться с помощью:

```
cqlsh ${cassandra_endpoint} -k ${keyspace_name} -f task_*.cql
```

Запуск PySpark скриптов в автоматическом режиме будет производиться с помощью:

```
spark-submit --conf
spark.cassandra.connection.host=${cassandra_endpoint} --packages
com.datastax.spark:spark-cassandra-connector_2.11:2.4.2 task_*.py
${keyspace_name}
```
[2]

Как создать DataFrame из таблицы Cassandra можно прочитать здесь

[2] Таким образом внутри PySpark скрипта рекомендуется использоваться sys.argv[1] для получения keyspace для записи данных

1.1. Описание данных

Данные о фильмах

movies.csv:

- Путь на кластере: полный датасет - **`/data/movielens/movies.csv`**
- Формат: текст
- В каждой строке находятся следующие поля, разделенные запятой:
    1. `movieId` - id фильма
    2. `title` - заголовок фильма (содержит год выпуска)
    3. `genres` - список жанров в виде строки слов, разделенных символом "|"

Пример:

```
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
```

Данные о рейтингах

ratings.csv:

- Путь на кластере: полный датасет - **`/data/movielens/ratings.csv`**
- Формат: текст
- В каждой строке находятся следующие поля, разделенные запятой:
    1. `userId` - id пользователя
    2. `movieId` - id фильма
    3. `rating`- оценка
    4. `timestamp` - время оценивания

Пример:

```
1,1441,4.0,945544871
1,1609,3.0,945544824
1,1961,3.0,945544871
```

1.2. Задание #1, Task ID: cassandra.create_keyspace

Создайте keyspace со стратегией репликации SimpleStrategy и фактором репликации 2. Назовите keyspace в соответствии с аргументом командной строки:

```
bash task_*_create_keyspace.sh ${cassandra_endpoint} ${keyspace_name}
```

1.3. Задание #2, Task ID: cassandra.movies_ddl

`movieid`	`int`
`title`	`text`
`year`	`int`
`genres`	`set<text>`

1. Создайте таблицу `movies` со схемой, приведенной выше. Таблица должна давать возможность выбрать всю информацию по конкретному названию фильма (title). Поскольку у одного фильма может быть несколько версий выхода на экран, то нужно предоставить возможность искать информацию по фильму и году одновременно (предлагается для извлечения года из названия использовать регулярное выражение "(\d+)"). Учтите, что в данных есть фильмы с одинаковым названием (title) и отличающимся id (movieid), следует разделять их как разные сущности. Заметьте, что жанры в исходном файле представляют из себя строку слов, разделенных символом вертикальной черты "|". В таблице в Cassandra жанры должны быть представлены типом set;
2. Напишите Spark SQL код, который будет считывать файл `movies.csv`, приводить DataFrame к правильному формату и запишет данные в таблицу `movies` в Cassandra. Обратите внимание, что в датасете могут быть ошибки, поэтому ошибки записи в Cassandra могут быть устранены предварительной очисткой данных[3];
3. Напишите запрос, который посчитает, сколько записей содержится в таблице (проверьте себя - должно получиться 37,968).

Названия CQL и PySpark скриптов:

- `task_*_movies_ddl.cql`
- `task_*_movies.py`
- `task_*_movies.cql + task_*_movies.out (STDOUT)`

[3] (1) предобработка quotes и trailing whitespaces в полях (до и после обработки сырых данных) (2) удаление строк без указания года и других характеристик (3) правильная обработка placeholder для данных, где не указаны жанры (найдите какой это placeholder)

1.4. Задание #3, Task ID: cassandra.movies_by_genre

1. Создайте таблицу `movies_by_genre`, которая отвечала бы на вопрос: получить все фильмы (movieid) в этом жанре (`genre`), отсортированные по убыванию года;
2. Напишите Spark SQL код, который будет брать данные из файла `movies.csv` и заполнит таблицу (каждое задание проверяется независимо от других, поэтому предполагается, что keyspace создан, но пустой);.
3. Напишите запрос, который подсчитает, сколько фильмов жанра Horror было снято в период с 1980 по 1990 год (включительно).

Названия CQL и PySpark скриптов:

- `task_*_movies_by_genre_ddl.cql`
- `task_*_movies_by_genre.py`
- `task_*_movies_by_genre.cql + task_*_movies_by_genre.out (STDOUT)`

1.5. Задание #4, Task ID: cassandra.movies_with_genre_index

1. Ту же информацию, что и в задании #3, можно получить из исходной таблицы movies, если создать вторичный индекс на столбце genres. Создайте таблицу `movies_with_genre_index` полностью аналогичную “movies” из задания #2. Создайте вторичный индекс (`genre_idx`) на столбце genres.
2. Напишите Spark SQL код, который будет брать данные из файла `movies.csv` и заполнит таблицу (каждое задание проверяется независимо от других, поэтому предполагается, что keyspace создан, но пустой).
3. Напишите запрос, который подсчитает, сколько фильмов жанра Horror было снято в период с 1980 по 1990 год (включительно). Заметьте, что такой запрос невозможно выполнить без опции `ALLOW FILTERING`. Внимательно прочитайте про эту опцию и запомните, что ее ни в коем случае нельзя использовать в продакшен запросах! Нужно создать другую схему данных.

Названия CQL и PySpark скриптов:

- `task_*_movies_with_genre_index_ddl.cql`
- `task_*_movies_with_genre_index.py`

- `task_*_movies_with_genre_index.cql + task_*_movies_with_genre_index.out (STDOUT)`

1.6. Задание #5, Task ID: cassandra.movies_by_genre_rating

1. Создайте таблицу `movies_by_genre_rating`, которая позволит отвечать на такой вопрос: вывести все фильмы (movieid) данного жанра в заданном диапазоне лет, отсортированные по убыванию среднего рейтинга фильма (поле rating, тип float). Таблица `movies_by_genre_rating` в поле rating должна хранить средний рейтинг фильма;
2. Напишите Spark SQL код, который будет брать данные из файлов `movies.csv` и `ratings.csv` и заполнит таблицу (каждое задание проверяется независимо от других, поэтому предполагается, что keyspace создан, но пустой).
3. Подсчитайте минимальный, средний и максимальный рейтинг у фильмов жанра Sci-Fi, вышедших в 21 веке (включая 2000 год) в таблице `movies_by_genre_rating`.

Названия CQL и PySpark скриптов:

- `task_*_movies_by_genre_rating_ddl.cql`
- `task_*_movies_by_genre_rating.py`
- `task_*_movies_by_genre_rating.cql + task_*_movies_by_genre_rating.out (STDOUT)`

2. Критерии оценивания

Балл за задачу складывается из:

- **100%** - правильное решение задачи
- **0%** - поддерживаемость и читаемость кода
    - в общем случае см. Clean Code и Google Python Style Guide
- **0%** - эффективность решения (такие как потребляемые CPU-ресурсы, скорость выполнения (в предположении свободного кластера)).

Discounts (скидки и другие акции):

- **100%** за плагиат в решениях (всем участникам процесса)
- **100%** за посылку решения после hard deadline
- **30%** за посылку решения в после soft deadline и до hard deadline
- **5%** за каждую дополнительную посылку в тестирующую систему (всего можно делать до 3-х посылок без штрафа):

Пример работы системы штрафов:

День	Посылка	Штраф
День 1	Посылка 1	Без штрафа
День 1	Посылка 2	Без штрафа
День 1	Посылка 3	Без штрафа
День 1	Посылка 4	-5%
День 2	Посылка 5	-5%
День 3	Посылка 6	-5%
Итоговый штраф: -15%		

Для подсчета финальной оценки **всегда** берется **последняя** оценка из Grader.

3. Инструкция по отправке задания

Перед отправкой задания оставьте, пожалуйста, отзыв о домашнем задании по ссылке: https://rebrand.ly/bdmade2022q2_feedback_hw. Это позволит нам скорректировать учебную нагрузку по следующим заданиям (в зависимости от того, сколько часов уходит на решение ДЗ), а также ответить на интересующие вопросы.

Оформление задания:

- Код задания (Short name): **HW09:NoSQL**
- Выполненное ДЗ запакуйте в архив **BD_MADE_2022_Q2**_<Surname>_<Name>_HW#**.zip**, например, для Алексея Драля -- **BD_MADE_2022_Q2**_Dral_Alexey_**HW09**.zip. (Проверяйте отсутствие пробелов и невидимых символов после копирования имени отсюда.[4]) Если ваше решение лежит в папке my_solution_folder, то для создания архива hw.zip на Linux и Mac OS, выполните команду[5]:
  - `zip -r hw.zip my_solution_folder/*`
- На Windows 7/8/10: необходимо выделить все содержимое директории my_solution_folder/ нажать правую кнопку мыши на одном из выделенных объектов, выбрать в открывшемся меню "Отправить >", затем "Сжатая ZIP-папка". Теперь можно переименовать архив.
- Перед проверкой убедитесь, что дерево вашего архива выглядит так:
  - **BD_MADE_2022_Q2_<Surname>_<Name>_HW09.zip**
  - **| ---- task_<Surname>_<Name>_create_keyspace.sh**
  - **| ---- task_<Surname>_<Name>_movies_ddl.cql**
  - **| ---- task_<Surname>_<Name>_movies.py**
  - **| ---- task_<Surname>_<Name>_movies.cql**
  - **| ---- task_<Surname>_<Name>_movies.out**
  - **| ---- task_<Surname>_<Name>_movies_by_genre_ddl.cql**
  - **| ---- task_<Surname>_<Name>_movies_by_genre.py**
  - **| ---- task_<Surname>_<Name>_movies_by_genre.cql**
  - **| ---- task_<Surname>_<Name>_movies_by_genre.out**
  - **| ---- task_<Surname>_<Name>_movies_with_genre_index_ddl.cql**
  - **| ---- task_<Surname>_<Name>_movies_with_genre_index.py**
  - **| ---- task_<Surname>_<Name>_movies_with_genre_index.cql**
  - **| ---- task_<Surname>_<Name>_movies_with_genre_index.out**
  - **| ---- task_<Surname>_<Name>_movies_by_genre_rating_ddl.cql**
  - **| ---- task_<Surname>_<Name>_movies_by_genre_rating.py**

[4] Онлайн инструмент для проверки: https://www.soscisurvey.de/tools/view-chars.php
[5] Флаг -r значит, что будет совершен рекурсивный обход по структуре директории

    - `| ---- task_<Surname>_<Name>_movies_by_genre_rating.cql`
    - `| ---- task_<Surname>_<Name>_movies_by_genre_rating.out`
    - При несовпадении дерева вашего архива с представленным деревом, ваше решение будет невозможно автоматически проверить, а значит, и оценить его.
- Для того, чтобы сдать задание, необходимо:
    - Зарегистрироваться и залогиниться в сервисе Everest
    - Перейти на страницу приложения: MADE BigData Grader
    - Выбрать вкладку Submit Job (если отображается иная).
    - Выбрать в качестве “Task” значение: **HW09:NoSQL**[6]
    - Загрузить в качестве “Task solution” файл с решением
    - В качестве Access Token указать тот, который был выслан по почте
- Если Вы видите надпись "You are not allowed to run this application" во вкладке Submit Job в Everest, то на данный момент сдача закрыта (нет доступных для сдачи домашних заданий, по техническим причинам или другое). Попробуйте, пожалуйста, еще раз через некоторое время. Если Вы еще ни разу не сдавали, у коллег сдача работает, но Вы видите такое сообщение, сообщите нам об этом.
- Ситуации:
  * система оценивания показывает оценку (Grade) < 0, а отчет (Grading report) не помогает решить проблему (пример помощи: в случае неправильно указанного Access Token система вернет -2 и информацию о том, что его нужно поправить);
  * показывает 0 и в отчете (Grading report) не указано, какие тесты не пройдены.
  Если Вы столкнулись с какой-то из них, присылайте ссылку на выполненное задание (Job) на почту с темой письма “Short name. ФИО.”. Например: “**HW09:NoSQL**. Иванов Иван Иванович.”
  Пример ссылки: https://everest.distcomp.org/jobs/67893456230000abc0123def
  Внимание: Если до дедлайна остается меньше суток, и Вы знаете (сами проверили или коллеги сообщили), что сдача решений сломана, обязательно сдайте свое решение и напишите письмо, как написано выше, чтобы мы видели, какое решение Вы имели до дедлайна и смогли его оценить.

Любые вопросы / комментарии / предложения можно писать в телеграм-канал курса или на почту bd_made2022q2@bigdatateam.org.

Всем удачи!

[6] Сервисный ID: cassandra.onsite_hw

4. FAQ (часто задаваемые вопросы)

Дополнительные источники
Документация по CQL: https://cassandra.apache.org/doc/latest/cassandra/cql/ Тест регулярных выражений: https://www.regexplanet.com/advanced/java/index.html

Что в отчете Grader означает проверка X ?
Как читать отчет: Для каждого теста • Raw_score - балл за конкретный тест. Может быть как бинарным (1\0), так и находиться в интервале от 0 до 1 • Score - Raw_score*weight (вес теста в общей оценке). Вес указан для каждого теста ниже Итоговая оценка: смотрите строку Score (сумма Score всех индивидуальных тестов) внизу отчета. **Правильность решения задачи:** **test_unzip_is_succesful** - ДЗ заархивировано в .zip архив и грейдер может его разархивировать, вес задания 0%; **test_pyspark_script_finished_succesfully** - код pyspark завершился успешно, вес задания 2% для каждого теста, 8% суммарно; **test_cassandra_table_has_expected_row_count** - число записей соответствует эталонному значению, вес задания 8% для `movies` и `movies_by_genre`, 4% для `movies_with_genre_index`, 20% суммарно; **test_create_keyspace_is_successful** - keyspace создан успешно, вес задания 2%; **test_keyspace_has_expected_replication_factor** - replication factor имеет верное значение, вес задания 9%;

test_keyspace_has_expected_replication_strategy - replication strategy имеет верное значение, вес задания 9%;

test_movies_by_genre_table_has_correct_fields - валидация колонок для таблицы `movies_by_genre`, вес задания 2%;

test_movies_by_genre_table_has_correct_primary_key - валидация первичного ключа для таблицы `movies_by_genre`, вес задания 4%;

test_movies_by_genre_table_has_correct_cluster_key - валидация cluster key для таблицы `movies_by_genre`, вес задания 4%;

test_movies_by_genre_rating_table_has_correct_fields - валидация колонок таблицы `movies_by_genre_rating`, вес задания 2%;

test_movies_by_genre_rating_table_has_correct_primary_key - валидация первичного ключа для таблицы `movies_by_genre_rating`, вес задания 4%;

test_movies_by_genre_rating_table_has_correct_cluster_key - валидация cluster key для таблицы `movies_by_genre_rating`, вес задания 4%;

test_movies_by_genre_rating_table_has_expected_ratings - сравнение значений рейтингов фильмов с эталонными значениями, вес задания 8%;

test_movies_table_has_correct_fields - валидация колонок для таблицы movies, вес задания 2%;

test_movies_table_has_correct_primary_key - валидация первичного ключа для таблицы `movies`, вес задания 4%;

test_movies_table_has_correct_cluster_key - валидация cluster key для таблицы `movies`, вес задания 4%;

test_movies_with_genre_index_table_has_correct_fields - валидация колонок для таблицы `movies_with_genre_index`, вес задания 2%;

test_movies_with_genre_index_table_has_correct_primary_key - валидация первичного ключа для таблицы `movies_with_genre_index`, вес задания 4%;

test_movies_with_genre_index_table_has_correct_cluster_key - валидация cluster key для таблицы `movies_with_genre_index`, вес задания 4%;

test_movies_with_genre_index_table_has_correct_index - валидация наличия индекса в таблице `movies_with_genre_index`, вес задания 4%;