FINISHED

Assignment 3: User Data Processing with Spark, HDFS, and Cassandra

Student Name: Zhang Zhuorui
Student ID: p147459
Tech Stack: Apache Spark | Cassandra | HDFS | Zeppelin Notebook

The goal of this project is to load user information (`u.user`) from the MovieLens dataset from HDFS, clean it using PySpark, and then store it into Apache Cassandra. User data analysis and visualizations will then be displayed using Zeppelin.

This Notebook is organized into the following sections:

1. Environment Initialization
2. Data Reading and Writing to Cassandra
3. User Data Analysis and Visualization
4. System Design Analysis (Courseware Extension)
5. Conclusion and Future Work
6. Areas for Improvement

Took 0 sec. Last updated by anonymous at June 19 2025, 9:43:29 PM. (outdated)

FINISHED

```
%pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Assignment3_UUser") \
    .master("local[*]") \
    .config("spark.cassandra.connection.host", "127.0.0.1") \
    .getOrCreate()

print("Spark Session 已启动")
```

```
Spark Session 已启动
```

Took 9 sec. Last updated by anonymous at June 19 2025, 9:33:39 PM.

FINISHED

Section 1 | Load User Data and Write to Cassandra Table `users`

We begin by reading the `u.user` file from HDFS and defining the Spark schema based on the field order and data types.
Next, the data is converted into a DataFrame and written into the `users` table in Cassandra using the Spark-Cassandra Connector.

Field definitions are as follows:

Field Definitions:

- **user_id**: Type: `int` , Description: Primary key, unique user identifier
- **age**: Type: `int` , Description: Age
- **gender**: Type: `string` , Description: Gender
- **occupation**: Type: `string` , Description: Occupation
- **zip**: Type: `string` , Description: Zip code

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:31 PM.

FINISHED

```
%pyspark
#1.1 读取 u.user 文件并显示前几行
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

# 定义 schema
user_schema = StructType([
    StructField("user_id", IntegerType(), True),
    StructField("age", IntegerType(), True),
    StructField("gender", StringType(), True),
    StructField("occupation", StringType(), True),
    StructField("zip", StringType(), True)
])

# 从 HDFS 加载数据（字段分隔符是 |
user_df = spark.read.csv(
    "hdfs:///user/maria_dev/ml-100k/u.user",
    sep="|",
    schema=user_schema
)
```

```
user_df.show(5, truncate=False)
```

```
+-------+---+------+----------+-----+
|user_id|age|gender|occupation|zip  |
+-------+---+------+----------+-----+
|1      |24 |M     |technician|85711|
|2      |53 |F     |other     |94043|
|3      |23 |M     |writer    |32067|
|4      |24 |M     |technician|43537|
|5      |33 |F     |other     |15213|
+-------+---+------+----------+-----+
only showing top 5 rows
```

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:40 PM.

FINISHED

```
%pyspark
# 1.2 写入 Cassandra 表 users
user_df.repartition(1).write \
    .format("org.apache.spark.sql.cassandra") \
    .option("table", "users") \
    .option("keyspace", "movielens") \
    .mode("append") \
    .save()
```

Took 1 sec. Last updated by anonymous at June 19 2025, 9:33:41 PM.

FINISHED

```
%pyspark
# 1.3 回读验证写入是否成功
spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(table="users", keyspace="movielens") \
    .load().show(10, truncate=False)
```

```
+-------+---+------+-------------+-----+
|user_id|age|gender|occupation   |zip  |
+-------+---+------+-------------+-----+
|79     |39 |F     |administrator|03755|
|210    |39 |M     |engineer     |03060|
|505    |27 |F     |other        |20657|
|16     |21 |M     |entertainment|10309|
|942    |48 |F     |librarian    |78209|
|402    |30 |M     |engineer     |95129|
|63     |31 |M     |marketing    |75240|
|768    |29 |M     |administrator|12866|
|725    |21 |M     |student      |91711|
|642    |18 |F     |student      |95521|
+-------+---+------+-------------+-----+
only showing top 10 rows
```

Took 1 sec. Last updated by anonymous at June 19 2025, 9:33:42 PM.

FINISHED

```
%md
- Table Name: `users`
- Primary Key Selection: `user_id` (single-field primary key)
- Read Scenarios: Query user details by `user_id`, or for JOIN operations in user profiling analysis
- Write Mode: Append, to prevent overwriting old records
- Corresponding to Week 9 Content: Schema-on-write principle in NoSQL, primary key must be defined
```

FINISHED

Section 2 | Load Rating Data and Write to Cassandra Table `ratings`

The file `u.data` contains 4 columns: User ID, Movie ID, Rating Value, and Timestamp (UNIX seconds).
To ensure the data can be used for time-based queries and sorting, we will convert the timestamp field to `timestamp` type before writing to Cassandra.

Table Schema Design:

- **movie_id**: Type: `int`, Description: Partition key
- **user_id**: Type: `int`, Description: Clustering key, for compound primary key
- **rating**: Type: `int`, Description: Rating value
- **ts**: Type: `timestamp`, Description: Rating timestamp (converted from UNIX timestamp)

The primary key is defined as `(movie_id, user_id)`, indicating that each user can have only one rating record per movie.

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:36 PM.

FINISHED

```
%pyspark
# 2.2 PySpark 读取 + 类型转换代码
from pyspark.sql.types import StructType, StructField, IntegerType, LongType
from pyspark.sql.functions import col, from_unixtime

# 定义 schema
```

```
schema_rating = StructType([
    StructField("user_id", IntegerType(), True),
    StructField("movie_id", IntegerType(), True),
    StructField("rating", IntegerType(), True),
    StructField("unix_ts", LongType(), True)
])

# 加载数据
rating_raw_df = spark.read.csv(
    "hdfs:///user/maria_dev/ml-100k/u.data",
    sep="\t",
    schema=schema_rating
)

# 转换 UNIX 时间戳为 timestamp
rating_df = rating_raw_df.withColumn(
    "ts", from_unixtime(col("unix_ts")).cast("timestamp")
).select("movie_id", "user_id", "rating", "ts")

rating_df.show(5, truncate=False)
```

```
+--------+-------+------+-------------------+
|movie_id|user_id|rating|ts                 |
+--------+-------+------+-------------------+
|242     |196    |3     |1997-12-04 15:55:49|
|302     |186    |3     |1998-04-04 19:22:22|
|377     |22     |1     |1997-11-07 07:18:36|
|51      |244    |2     |1997-11-27 05:02:03|
|346     |166    |1     |1998-02-02 05:33:16|
+--------+-------+------+-------------------+
only showing top 5 rows
```

Took 2 sec. Last updated by anonymous at June 19 2025, 9:33:43 PM.

FINISHED

```
%pyspark
# 2.3 写入 Cassandra 表 ratings
rating_df.repartition(1).write \
    .format("org.apache.spark.sql.cassandra") \
    .option("table", "ratings") \
    .option("keyspace", "movielens") \
    .mode("append") \
    .save()
```

Took 1 min 3 sec. Last updated by anonymous at June 19 2025, 9:34:45 PM.

FINISHED

```
%pyspark
# 2.4 验证写入成功
spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(table="ratings", keyspace="movielens") \
    .load().show(10, truncate=False)
```

```
+--------+-------+------+-------------------+
|movie_id|user_id|rating|ts                 |
+--------+-------+------+-------------------+
|1201    |90     |5     |1998-03-31 22:34:47|
|79      |1      |4     |1997-09-24 03:47:45|
|79      |5      |3     |1997-09-30 16:11:35|
|79      |6      |3     |1997-12-31 20:39:07|
|79      |7      |4     |1998-03-31 13:51:01|
|79      |8      |4     |1997-11-12 19:18:06|
|79      |11     |4     |1998-04-06 23:36:23|
|79      |13     |3     |1997-12-14 22:49:06|
|79      |16     |5     |1997-10-24 21:05:22|
|79      |18     |4     |1997-11-21 16:57:30|
+--------+-------+------+-------------------+
only showing top 10 rows
```

Took 1 min 4 sec. Last updated by anonymous at June 19 2025, 9:34:47 PM.

FINISHED

Primary Key Design Explanation:

- Partition Key: `movie_id`, to facilitate aggregating ratings for each movie
- Clustering Key: `user_id`, to enable quick lookup of whether a user has rated a specific movie
- Using a compound primary key `(movie_id, user_id)` prevents users from rating the same movie multiple times

Timestamp Field Conversion:

- Use `from_unixtime()` to convert `int` to Spark-supported `timestamp`
- Automatically maps to `timestamp` type when writing to Cassandra, facilitating subsequent time analysis

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:38 PM.

FINISHED

Section 3 | Load Movie Information and Write to Cassandra Table `movies`

Section 3 | Load Movie Information and Write to Cassandra Table `movies`

The file `u.item` contains detailed movie information, including:

- `movie_id` (Primary Key)
- `title` (Movie Title)
- 19 boolean flags indicating whether the movie belongs to corresponding genres (e.g., Action, Comedy)

To improve readability and facilitate subsequent analysis, we will convert the genres into a **set type**, storing multiple genres for each movie.

Field Definitions:

- **movie_id**: Type: `int`, Description: Primary key
- **title**: Type: `text`, Description: Movie title
- **genres**: Type: `set<text>`, Description: Set of genres (e.g., `{'Action', 'Drama'}`)

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:38 PM.

FINISHED

```
%pyspark
# 3.2 加载 u.genre 映射表
# 加载 genre 映射编号 → 类型名（例如 0 → Action）
genre_map = (
    spark.read.text("hdfs:///user/maria_dev/ml-100k/u.genre")
    .filter("value != ''")
    .rdd.map(lambda r: r[0].split("|"))
    .map(lambda kv: (int(kv[1]), kv[0]))
    .collectAsMap()
)

print(genre_map)  # 可查看 genre 索引和名称映射关系
```

```
{0: u'unknown', 1: u'Action', 2: u'Adventure', 3: u'Animation', 4: u"Children's", 5: u'Comedy', 6: u'Crime', 7: u'Documentary', 8: u'Drama', 9: u'Fantasy', 1
0: u'Film-Noir', 11: u'Horror', 12: u'Musical', 13: u'Mystery', 14: u'Romance', 15: u'Sci-Fi', 16: u'Thriller', 17: u'War', 18: u'Western'}
```

Took 2 sec. Last updated by anonymous at June 19 2025, 9:34:47 PM.

FINISHED

```
%pyspark
# 3.3 加载 u.item 文件并解析 genre 位
from pyspark.sql.types import *

# genre 列（注意此处不使用 f-string，改为 format 兼容 Zeppelin）
genre_fields = [StructField("g{}".format(i), IntegerType(), True) for i in range(19)]

# 构造 schema（前 5 列 + 19 个 genre 布尔列）
prefix_fields = [
    StructField("movie_id", IntegerType(), True),
    StructField("title", StringType(), True),
    StructField("release_date", StringType(), True),
    StructField("video_release_date", StringType(), True),
    StructField("imdb_url", StringType(), True)
]

schema_item = StructType(prefix_fields + genre_fields)

item_raw = spark.read.csv(
    "hdfs:///user/maria_dev/ml-100k/u.item",
    sep="|",
    schema=schema_item
)

item_raw.show(5, truncate=False)
```

```
+--------+-----------------+------------+-----------------+-----------------------------------------------------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|movie_id|title            |release_date|video_release_date|imdb_url                                             |g0 |g1 |g2 |g3 |g4 |g5 |g6 |g7 |g8 |g9 |g10|g11|g12|g13|g14|g15|g16|g17|g18|
+--------+-----------------+------------+-----------------+-----------------------------------------------------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|1       |Toy Story (1995) |01-Jan-1995 |null             |http://us.imdb.com/M/title-exact?Toy%20Story%20(1995) |0  |0  |0  |1  |1  |1  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |
|2       |GoldenEye (1995) |01-Jan-1995 |null             |http://us.imdb.com/M/title-exact?GoldenEye%20(1995)   |0  |1  |1  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |1  |0  |0  |0  |0  |
|3       |Four Rooms (1995)|01-Jan-1995 |null             |http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)|0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |1  |0  |0  |0  |0  |
|4       |Get Shorty (1995)|01-Jan-1995 |null             |http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)|0  |1  |0  |0  |0  |1  |0  |0  |1  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |
|5       |Copycat (1995)   |01-Jan-1995 |null             |http://us.imdb.com/M/title-exact?Copycat%20(1995)     |0  |0  |0  |0  |0  |0  |1  |0  |1  |0  |0  |0  |0  |0  |1  |0  |0  |0  |0  |
+--------+-----------------+------------+-----------------+-----------------------------------------------------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Took 0 sec. Last updated by anonymous at June 19 2025, 9:34:47 PM.

FINISHED

```
%pyspark
# 3.4 转换 genre 位 → set<text>
# 定义转换函数
def extract_genres(row):
    genres = [genre_map[i] for i in range(19) if row["g{}".format(i)] == 1]
    return (row["movie_id"], row["title"], genres)  # 不建议用 set，DataFrame不支持
```

```
# RDD 转换为 (movie_id, title, [genre1, genre2, ...])
movie_rdd = item_raw.rdd.map(extract_genres)

# 转换为 DataFrame
movie_df = spark.createDataFrame(movie_rdd, ["movie_id", "title", "genres"])

# 展示前5行
movie_df.show(5, truncate=False)
```

```
+--------+----------------+------------------------------+
|movie_id|title           |genres                        |
+--------+----------------+------------------------------+
|1       |Toy Story (1995) |[Animation, Children's, Comedy]|
|2       |GoldenEye (1995) |[Action, Adventure, Thriller]  |
|3       |Four Rooms (1995)|[Thriller]                     |
|4       |Get Shorty (1995)|[Action, Comedy, Drama]        |
|5       |Copycat (1995)   |[Crime, Drama, Thriller]       |
+--------+----------------+------------------------------+
only showing top 5 rows
```

Took 1 sec. Last updated by anonymous at June 19 2025, 9:34:48 PM.

FINISHED

```
%pyspark
# 3.5 写入 Cassandra 表 movies
movie_df.repartition(1).write \
    .format("org.apache.spark.sql.cassandra") \
    .option("table", "movies") \
    .option("keyspace", "movielens") \
    .mode("append") \
    .save()
```

Took 2 sec. Last updated by anonymous at June 19 2025, 9:34:50 PM.

FINISHED

```
%pyspark
# 3.6 验证写入是否成功
spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(table="movies", keyspace="movielens") \
    .load().show(10, truncate=False)
```

```
+--------+-------------------+----------------------------------------+
|movie_id|genres             |title                                   |
+--------+-------------------+----------------------------------------+
|1201    |[Documentary]      |Marlene Dietrich: Shadow and Light (1996) |
|79      |[Action, Thriller] |Fugitive, The (1993)                    |
|210     |[Action, Adventure]|Indiana Jones and the Last Crusade (1989) |
|1289    |[Romance]          |Jack and Sarah (1995)                   |
|505     |[Mystery, Thriller]|Dial M for Murder (1954)                |
|1611    |[Comedy]           |Intimate Relations (1996)               |
|16      |[Comedy, Romance]  |French Twist (Gazon maudit) (1995)      |
|1061    |[Comedy, Drama]    |Evening Star, The (1996)                |
|1211    |[Drama, Romance]   |Blue Sky (1994)                         |
|1468    |[Drama]            |Cure, The (1995)                        |
+--------+-------------------+----------------------------------------+
only showing top 10 rows
```

Took 3 sec. Last updated by anonymous at June 19 2025, 9:34:51 PM.

FINISHED

- Each movie can correspond to multiple genres, thus a `set<text>` type is used for storage.
- The `genres` field parsing is based on the `u.genre` mapping, avoiding direct use of boolean bit numbering.
- Can be JOINed with the `ratings` table via `movie_id` for convenient analysis of rating performance across different genres.

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:41 PM.

FINISHED

Section 4 | Answering Five Specific Query Questions (Assignment i–v)

In this section, we will use Spark SQL to answer the following five questions by querying the three previously written tables:

1. The average rating for each movie
2. The top 10 movies by average rating (including titles)
3. The favorite movie genre of each active user
4. All users under the age of 20
5. Users aged between 30 and 40 whose occupation is “scientist”

To simplify the analysis, we first register the three tables as temporary views (TempView).

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:42 PM.

FINISHED

```
%pyspark
# 4.2 注册三张 Cassandra 表为视图
```

```
spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(table="users", keyspace="movielens") \
    .load().createOrReplaceTempView("users")

spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(table="ratings", keyspace="movielens") \
    .load().createOrReplaceTempView("ratings")

spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(table="movies", keyspace="movielens") \
    .load().createOrReplaceTempView("movies")
```

Took 3 sec. Last updated by anonymous at June 19 2025, 9:34:54 PM.

FINISHED

(i) Average Rating for Each Movie

- This query uses the `ratings` table, grouping by `movie_id` and calculating `AVG(rating)`.
- The results can be used for subsequent filtering of highly-rated movies.

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:43 PM.

FINISHED

```
%pyspark
# (i) 每部电影的平均评分
spark.sql("""
SELECT r.movie_id, m.title, ROUND(AVG(r.rating), 2) AS avg_rating
FROM ratings r
JOIN movies m ON r.movie_id = m.movie_id
GROUP BY r.movie_id, m.title
ORDER BY r.movie_id
""").show(10)
```

```
+--------+--------------------+----------+
|movie_id|               title|avg_rating|
+--------+--------------------+----------+
|       1|    Toy Story (1995)|      3.88|
|       2|    GoldenEye (1995)|      3.21|
|       3|   Four Rooms (1995)|      3.03|
|       4|   Get Shorty (1995)|      3.55|
|       5|      Copycat (1995)|       3.3|
|       6|Shanghai Triad (Y...|      3.58|
|       7|Twelve Monkeys (1...|       3.8|
|       8|         Babe (1995)|       4.0|
|       9|Dead Man Walking ...|       3.9|
|      10|  Richard III (1995)|      3.83|
+--------+--------------------+----------+
only showing top 10 rows
```

Took 23 sec. Last updated by anonymous at June 19 2025, 9:35:14 PM.

FINISHED

(ii) Top Ten Movies with Highest Average Rating

- Join `ratings` and `movies` tables to get movie titles by movie ID.
- Sort by `avg_rating DESC` to select the top 10 movies with the highest ratings.

Took 0 sec. Last updated by anonymous at June 19 2025, 9:41:07 PM. (outdated)

FINISHED

```
%pyspark
# (ii) 平均评分最高的前十部影片（含标题）
spark.sql("""
SELECT r.movie_id, m.title, ROUND(AVG(r.rating), 2) AS avg_rating
FROM ratings r
JOIN movies m ON r.movie_id = m.movie_id
GROUP BY r.movie_id, m.title
ORDER BY avg_rating DESC
""").show(n = 10, truncate = False)
```

```
+--------+-------------------------------------------------+----------+
|movie_id|title                                            |avg_rating|
+--------+-------------------------------------------------+----------+
|1201    |Marlene Dietrich: Shadow and Light (1996)        |5.0       |
|1122    |They Made Me a Criminal (1939)                   |5.0       |
|1189    |Prefontaine (1997)                               |5.0       |
|814     |Great Day in Harlem, A (1994)                    |5.0       |
|1293    |Star Kid (1997)                                  |5.0       |
|1599    |Someone Else's America (1995)                    |5.0       |
|1467    |Saint of Fort Washington, The (1993)             |5.0       |
|1536    |Aiqing wansui (1994)                             |5.0       |
|1653    |Entertaining Angels: The Dorothy Day Story (1996)|5.0       |
|1500    |Santa with Muscles (1996)                        |5.0       |
```

```
+--------+--------------------------------------------------+----------+
only showing top 10 rows
```

Took 24 sec. Last updated by anonymous at June 19 2025, 9:35:18 PM.

FINISHED

(iii) For each user with at least 50 ratings, identify their favorite movie genre

- First, filter the "active users" group (those with ≥ 50 ratings).
- Then, extract the movie genres rated by these users through a JOIN operation.
- Use `explode()` to flatten the collection and count the most frequently occurring genres.

Took 0 sec. Last updated by anonymous at June 19 2025, 9:33:48 PM.

FINISHED

```
%pyspark
# （iii）每个评分≥50次的用户最喜欢的电影类型
spark.sql("""
WITH active_users AS (
    SELECT user_id
    FROM ratings
    GROUP BY user_id
    HAVING COUNT(*) >= 50
),
exploded_genres AS (
    SELECT r.user_id, genre
    FROM ratings r
    JOIN active_users a ON r.user_id = a.user_id
    JOIN movies m ON m.movie_id = r.movie_id
    LATERAL VIEW explode(m.genres) AS genre
),
genre_counts AS (
    SELECT user_id, genre, COUNT(*) AS count
    FROM exploded_genres
    GROUP BY user_id, genre
),
ranked_genres AS (
    SELECT *,
        ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY count DESC) AS rank
    FROM genre_counts
)
SELECT g.user_id, u.gender, u.occupation, g.genre, g.count
FROM ranked_genres g
JOIN users u ON g.user_id = u.user_id
WHERE rank = 1
ORDER BY g.count DESC

""").show(10, truncate=False)
```

```
+-------+------+----------+------+-----+
|user_id|gender|occupation|genre |count|
+-------+------+----------+------+-----+
|655    |F     |healthcare|Drama |410  |
|405    |F     |healthcare|Drama |309  |
|537    |M     |engineer  |Drama |251  |
|450    |F     |educator  |Drama |237  |
|13     |M     |educator  |Drama |218  |
|234    |M     |retired   |Drama |213  |
|416    |F     |student   |Drama |212  |
|279    |M     |programmer|Comedy|211  |
|201    |M     |writer    |Drama |196  |
|393    |M     |student   |Comedy|191  |
+-------+------+----------+------+-----+
only showing top 10 rows
```

Took 20 sec. Last updated by anonymous at June 19 2025, 9:35:34 PM.

FINISHED

(iv) All users under the age of 20

- Simply filter the `users` table, selecting records where the age field is less than 20.

Took 0 sec. Last updated by anonymous at June 19 2025, 9:39:32 PM.

FINISHED

```
%pyspark

spark.sql("""
SELECT * FROM users
WHERE age < 20
""").show(n = 10, truncate = False)
```

```
+-------+---+------+-------------+-----+
|user_id|age|gender|occupation   |zip  |
+-------+---+------+-------------+-----+
|642    |18 |F     |student      |95521|
|588    |18 |F     |student      |93063|
|30     |7  |M     |student      |55436|
```

```
|528    |18 |M      |student      |55104|
|674    |13 |F      |student      |55337|
|375    |17 |M      |entertainment|37777|
|851    |18 |M      |other        |29646|
|859    |18 |F      |other        |06492|
|813    |14 |F      |student      |02136|
|52     |18 |F      |student      |55105|
+-------+---+-------+-------------+-----+
only showing top 10 rows
```

Took 15 sec. Last updated by anonymous at June 19 2025, 9:35:34 PM.

FINISHED

(v) Users aged between 30 and 40 whose occupation is “scientist”

- Use `BETWEEN` and `=` to combine filtering conditions.

Took 0 sec. Last updated by anonymous at June 19 2025, 9:34:50 PM.

FINISHED

```
%pyspark
# (v) 年龄在 30-40 岁之间且职业为 scientist 的用户
spark.sql("""
SELECT * FROM users
WHERE age BETWEEN 30 AND 40 AND occupation = 'scientist'
""").show(n = 10, truncate=False)
```

```
+-------+---+------+----------+-----+
|user_id|age|gender|occupation|zip  |
+-------+---+------+----------+-----+
|730    |31 |F     |scientist |32114|
|74     |39 |M     |scientist |T8H1N|
|183    |33 |M     |scientist |27708|
|107    |39 |M     |scientist |60466|
|918    |40 |M     |scientist |70116|
|337    |37 |M     |scientist |10522|
|272    |33 |M     |scientist |53706|
|430    |38 |M     |scientist |98199|
|643    |39 |M     |scientist |55122|
|543    |33 |M     |scientist |95123|
+-------+---+------+----------+-----+
only showing top 10 rows
```

Took 1 sec. Last updated by anonymous at June 19 2025, 9:35:35 PM.

FINISHED

Section 5 | Summary and Future Work

roject Summary

In this Assignment 3, I have completed the following key tasks:

1. Used Apache Zeppelin to build an interactive Notebook, achieving an interpretable and reproducible data processing workflow.
2. Used PySpark to load three sub-files of the MovieLens dataset (u.user, u.data, u.item) from HDFS.
3. Combined Schema definition and cleaning logic to structure the data into DataFrames.
4. Utilized the Spark-Cassandra Connector to write three types of data (users, ratings, movies) into Cassandra tables (users, ratings, movies).
5. Designed reasonable primary keys and data types, particularly employing a composite primary key `(movie_id, user_id)` in `ratings`.
6. Implemented five core query tasks, covering single-table aggregation, JOINs, multi-level filtering, set unnesting, and other complex analytical operations.
7. Supplemented each analysis step with chart displays and Markdown documentation to ensure a clear and logically complete Notebook structure.

Course Knowledge Application Points

Concept	Application Location
CAP Theory	Selected Cassandra (AP model) for high-availability distributed writes
Hadoop HDFS	Raw data loading paths based on HDFS storage structure
ETL Process	Extract (HDFS) → Transform (Spark cleansing) → Load (Cassandra)
Distributed System Challenges	Used `repartition(1)` to control concurrent writes, solving the risk of Zeppelin write failures
NoSQL Data Modeling	Used `set<text>` collection type to store movie genres, reasonably designed primary keys for optimized query performance

---

Took 0 sec. Last updated by anonymous at June 19 2025, 9:40:51 PM.

FINISHED

```
%pyspark
```