Spark SQL Experiment: E-commerce User Behavior Analysis

**1. Experimental objectives**

1. Master the core syntax of Spark SQL ('SELECT'/'GROUP BY'/'WINDOW'/conditional aggregation, etc.) and understand the logic of structured data processing.

2. Compare the differences between Spark RDD and Spark SQL, and experience the simplicity advantages of SQL for "structured analysis scenarios".

3. Based on e-commerce user behavior data, use Spark SQL to complete basic statistics and advanced analysis, and output actionable business conclusions.

**2. Environment preparation and data loading**

1. Environment Preparation (Reuse the original foundation, supplement SQL configuration)

- Spark, Python environment installed.

- Make sure the 'pyspark' library is installed (consistent with the Spark version, e.g. 'pip install pyspark==3.0.1').

2. Spark SQL Data Loading and View Creation (Core Steps)

Loading CSV data via Spark SQL and creating **temporary views** (for subsequent SQL queries) is all about defining strict schemas to improve efficiency

**3. Core tasks (Spark SQL implementation)**

All tasks are executed via 'spark.sql ('SQL statement'), and the results are saved to the 'output' folder, which supports csv format.

**Fundamental Analysis (Required)**

**Task 1.1:** Count the total number of 4 behaviors + calculate the purchase conversion rate

**Task objective**: Clarify the distribution of each behavior (click/favorite/add/purchase) and quantify the core conversion efficiency of "click→purchase".

**Pseudocode**:

1. From the temporary view user\_behavior\_view, group by action field, use COUNT(\*) to count the total number of times of each behavior, sort in descending order of the total number of times, and create a temporary view action\_count\_view;

2. From the action\_count\_view, extract the total number of action = 'buy' and the total number of action = 'click' respectively through sub-query, and calculate the purchase conversion rate by "(number of purchases / clicks)×100", keeping 2 decimal places;

3. Save the action\_count\_view result to the specified path, and the terminal outputs the conversion rate result.

**Task 1.2:** Analyze the popularity of product categories (in descending order of clicks).

**Task objective**: Identify the most popular product categories and provide a basis for "category operation".

**Pseudocode**:

1. From the temporary view user\_behavior\_view, filter the behavior data for action = 'click';
2. Group by category field, use COUNT(\*) to count the number of clicks in each category, sort them in descending order of clicks, and create a temporary view category\_click\_view;
3. Save the category\_click\_view results to the specified path, and you can output the first N data to view popular categories.

**Task 1.3:** Analyze the active period of the user (the number of behaviors by hour).

**Task objectives**: Identify the peak activity of users in the day, and guide "limited-time promotions" and "customer service scheduling".

**Pseudocode**:

1. From the temp view user\_behavior\_view, use a time function to convert the timestamp (Unix timestamp) to an 'hour' format (e.g., FROM\_UNIXTIME(timestamp, 'HH'));
2. Group according to the converted "Hours" field, use COUNT(\*) to count the total number of actions for each hour, sort them in ascending order of hours, and create a temporary view hourly\_behavior\_view.
3. Save the hourly\_behavior\_view results to the specified path, and filter the time period with the highest number of behaviors as the active peak.

**Advanced analysis (selection, ability improvement)**

**Task 2.1:** Identify high-value users (Top 3 purchasing users).

**Mission objective**: Identify the 3 users with the most purchases and provide a list for "high-value user operations" (e.g., exclusive benefits).

**Pseudocode**:

1. From the temporary view user\_behavior\_view, filter the behavior data for action = 'buy';
2. Group by user\_id field, use COUNT(\*) to count the number of purchases of each user, sorted in descending order of the number of purchases;
3. Take the first 3 sorted data, create a temporary view top3\_buy\_user\_view, and save the results to the specified path.

**Task 2.2:** Conversion funnel analysis (click → favorite→add-on→ purchase).

**Task objective**: Quantify the conversion loss of each link of the funnel, and locate the "links with serious churn" (such as collection→ the conversion rate of add-on purchases is low, and the product detail page needs to be optimized).

**Pseudocode**:

1. Behavior path marking: From the user behavior view, group by user ID and product ID, sort the behavior of each user-product by timestamp, and mark the behavior order with window functions (such as the first behavior, the second behavior, etc.), and create a temporary view 'user\_item\_behavior\_path', including fields: user\_id, item\_id, action, timestamp, action\_order.

2. Funnel Effectiveness Filtering:

From 'user\_item\_behavior\_path', filter the data with the behavior type click, collect, cart, buy,

Only the continuous behavior of "click(action\_order=1)→collect(action\_order=2)→cart(action\_order=3)→buy(action\_order=4)" is retained,

Mark whether each behavior is a valid funnel (is\_valid\_funnel=1 means valid),

Create a temporary view 'valid\_funnel\_behavior'.

3. Count the number of unique users in each link (only effective funnel behavior):

From 'valid\_funnel\_behavior', the statistics are calculated separately by conditional aggregation:

Number of unique users in the click link: click\_user = the number of distinct user\_id with action='click' and is\_valid\_funnel=1;

Number of unique users in the collection link: collect\_user = number of distinct user\_id with action='collect' and is\_valid\_funnel=1;

Number of unique users in the add-on link: cart\_user = number of distinct user\_id with action='cart' and is\_valid\_funnel=1;

Number of unique users in the purchase process: buy\_user = number of distinct user\_id with action='buy' and is\_valid\_funnel=1;

Create a temporary view 'funnel\_user\_count\_view'.

4. Calculate the conversion rate of each link:

From 'funnel\_user\_count\_view', calculate the conversion rate of each link:

Click → collection conversion rate: click\_to\_collect = (collect\_user / click\_user) \* 100, keep 2 decimal places;

Conversion rate of add-on → collection: collect\_to\_cart = (cart\_user / collect\_user) \* 100, keep 2 decimal places;

Conversion rate of add-on → purchase: cart\_to\_buy = (buy\_user / cart\_user) \* 100, with 2 decimal places;

Overall conversion rate: overall\_conversion = (buy\_user / click\_user) \* 100, keep 2 decimal places;

Create a temporary view 'funnel\_conversion\_view'.

5. Save and Output Results:

Save the result of 'funnel\_conversion\_view' as a CSV file to a specified path (e.g. output/sql\_advanced\_funnel);

The terminal outputs the conversion rate results of each link.

**Task 2.3:** Analysis of the repurchase rate of goods

**Task objective**: Count the repurchase rate of each product (the number of users who have purchased ≥ 2 times/the total number of users who have purchased it), and identify "loyal products with a high repurchase rate".

**Pseudocode**:

1. From the temporary view user\_behavior\_view, filter the behavior data of action = 'buy', group by item\_id and user\_id, and use COUNT(\*) to count the number of purchases of each product by each user to create a temporary view item\_user\_buy\_count\_view;
2. From the item\_user\_buy\_count\_view, group by item\_id:

Count the total number of users who purchase the product: COUNT (DISTINCT user\_id);

Statistics on the number of repeat users (number of purchases≥ 2): COUNT(DISTINCT CASE WHEN buy\_times >= 2 THEN user\_id END);

Calculate the repurchase rate according to "(number of repurchase users / total number of purchasing users)×100", keep 2 decimal places, sort in descending order of repurchase rate, and create a temporary view item\_repurchase\_view;

1. Save the item\_repurchase\_view results to the specified path.

**Task 2.4:** User next-day retention

**Task objective**: Calculate the proportion of "active users on the same day are still active the next day" to measure the "retention ability" of the product to users.

**Pseudocode**:

1. From the temporary view user\_behavior\_view, extract the user\_id and the "timestamp converted date" (such as DATE(FROM\_UNIXTIME(timestamp))), and after deduplication, obtain the user's daily active record, and create a temporary view user\_daily\_active\_view.
2. From the user\_daily\_active\_view, use the LAG() window function (sorted by user\_id partitions, active\_date) to get each user's "active date of the previous day" to create a temporary view user\_prev\_active\_view;
3. From the user\_prev\_active\_view, group by active\_date:

Count the number of active users on the day: COUNT (DISTINCT user\_id);

Count the number of users retained on the next day (active the previous day and still active on the same day, i.e., DATEDIFF(active\_date, prev\_active\_date) = 1);

Calculate the next day's retention rate by clicking "(Number of Retained Users / Number of Active Users on the Day)×100", keep 2 decimal places, and create a temporary view user\_retention\_view.

1. Save the user\_retention\_view results to the specified path.

**Task 2.5:** Analyze and count the most popular videos on the main site

Mission Objectives Statistics Top 20 Popular Videos

### Data Sources:[net](https://www.imooc.com/)The site access log uses access\_20161111.log as the initial data, the data is the date, time, URL, traffic, IP address, where the URL is like this”http://www.imooc.com/video/4500” where video is the video, followed by the video ID.

Expand:

Create a data table based on access\_20161111.log and compare the time it takes to complete the task with the native SQL query statement and the time spent using SparkSQL.

Study program address: https://github.com/ptyadana/SQL-Data-Analysis-and-Visualization-Projects