Cheetah Loyalty Phoenix Query

Developer Guide

Version	Date	Description	Reviewed / Approved by
1.0	August 2019	Initial release	Cheetah Digital Product Management
1.1	September 2021	Added version history	Cheetah Digital Product Management

Cheetah Digital, Inc., 72 W Adams St 8th floor, Chicago, IL 60603

Printed in the United States of America

Table of Contents

Introduction

This document describes the process of creating Analytics Queries using Phoenix syntax in the Cheetah Loyalty admin console.

Creating Queries

Phoenix queries can be created and executed on Cheetah Loyalty console via **Analytics Queries** screen. To create a new Phoenix query:

1. On the **Analytics** Screen, expand the Settings menu on the left panel and click Queries menu item.
2. On **Analytics Queries** screen, enter the title/name of the query to be created and click the **Create** button.
3. On the **Edit Bigdata Query** modal form, set Execution type to **Phoenix (enhanced)** and click save.
4. Find the query in the **Analytics Queries** list page, and click into the record to view the details and configure the Phoenix query.
5. To configure the query, click the Edit button on the query **Definition** detail tab, configure the desired query and save the changes.
6. To view the query results, select the query **Results** detail tab and click the Refresh button.

IMPORTANT - Running sub-optimized queries in the Production environment can result in significant performance degradation that can impact program user experience so only author new queries in pre-production/staging environments. Move queries to Production only once they've been thoroughly tested in a pre-production environment.

Tables

1. MEMBER_TABLE

Contains information about members, this includes basic information such as member_id, name, birthdate, gender, etc. Address, All-time metrics, and Membership summary is also available in this table.

Primary Key

member_id

Fields/Columns

For the fields/column names and Data type, you may refer to the internal names of all **Member Attributes**, **Member Preferences** and **Member Metrics** on the **Program** menu.

Sample Queries

Basic

Get all the available columns of a certain member:

```
SELECT  *
FROM  MEMBER_TABLE
WHERE  member_id  =  'member01'
```

Get the name, birthdate, and gender of a certain member:

```
SELECT  first_name  ||  last_name  AS  name,
        birthdate,
        gender
FROM  MEMBER_TABLE
WHERE  member_id  =  'member01'
```

Member Preferences:

```
SELECT interests
FROM MEMBER_TABLE
WHERE member_id = 'member01'
```

Aggregates and Grouping

Group members by Source

```
SELECT CASE
WHEN source is NULL THEN 'Undefined'
ELSE source
END AS "Source",
```

```
    count(1) AS "Member Count"
FROM MEMBER_TABLE
GROUP BY source
ORDER BY "Member Count"
desc
```

Group members by age group

```
SELECT  CASE
            WHEN  age  (birthdate,  'year')  >= 65 THEN  '65+'
            WHEN  age  (birthdate,  'year')  >= 45 THEN  '45-64'
            WHEN  age  (birthdate,  'year')  >= 25 THEN  '25-44'
            WHEN  age  (birthdate,  'year')  >= 15 THEN  '15-24'
            WHEN  age  (birthdate,  'year')  > 0 THEN  '<15'
            ELSE  'Others'
      END  AS  "Age Group",
count(*)
FROM  MEMBER_TABLE
GROUP  BY  "Age Group"
```

2. ACTIVITY_TABLE

Activities are user interactions like purchases, reward redemptions, challenge responses, email delivery information, profile and preference updates, and so on.

Primary Keys (Composite Keys)

sl_activity_ts, sl_type, sl_member_id, sl_context, sl_ext_id, sl_id, sl_subtype

Fields/Columns

To learn more about available column names, see the Program > Activity Types screen in the Cheetah admin console. The internal name of each activity type corresponds to an **sl_type** value that can be used in analytics queries. Additionally, each activity type specifies a list of available attributes that can be returned for that type.

Header Attributes
Summary-level attributes supported by the activity_type.

System Attributes
System-generated attributes available for all activity types.

Sample Queries

Total activities per earn type:

```
SELECT  earn_type  AS  "Earn Type",
      Count(1)  AS  "Activity Count"
```

```
FROM  ACTIVITY_TABLE
WHERE  in_period(sl_activity_ts,  'alltime')  =  true
GROUP  BY  earn_type
```

Activity type period summary:

```
SELECT  sl_type  AS  "Activity Type",
       Count(*)  AS  "Activity Count",
       Count(DISTINCT  sl_member_id)  AS  "Member Count",
       Coalesce(SUM(CASE  WHEN  Lower(earn_type)  =  'earn'  THEN  point
END), 0)  AS  "Earn",
       Coalesce(SUM(CASE  WHEN  Lower(earn_type)  =  'expire'  THEN  point
END), 0)  AS  "Expire",
       Coalesce(SUM(CASE  WHEN  Lower(earn_type)  =  'redeem'  THEN  point
END), 0)  AS  "Redeem"
FROM  ACTIVITY_TABLE
WHERE  in_period(sl_activity_ts,  'alltime')  =  TRUE
GROUP  BY  sl_type
```

3. ACTIVITY_ITEM_TABLE

For activity types with line item details, the data is available in the ACTIVITY_ITEM_TABLE. One example activity type is Update Member Attribute activity - each attribute that is updated will have its entry in the ACTIVITY_ITEM_TABLE. Another use case is on Purchase activity, each item that is included in a transaction will have an individual entry in this table.

Primary Keys (Composite Keys)

sl_activity_ts, sl_type, sl_item_name, sl_item_category,sl_member_id, sl_ext_id, sl_id, sl_context, sl_item_id, sl_subtype

Fields/Columns

Not all activity types have line items. To learn more about available column names, see the Program > Activity Types screen in the Cheetah admin console. The Line Item Attributes includes all available line items for that specific activity. A blank list indicates that the activity does not have any line item attributes.

Sample Queries

Count of members who filled-out their First Name in 1 month.

```
SELECT  count(DISTINCT  sl_member_id)
FROM  ACTIVITY_ITEM_TABLE
WHERE  sl_type = 'sl_member_attribute'
       AND  sl_attribute = 'first_name'
       AND  sl_attribute_value  IS NOT NULL
       AND  sl_prev_value  IS NULL
```

```
        AND  in_period(sl_activity_ts,  'last1m')  =  TRUE
```

Top 10 Styles in 1 Month

```
SELECT style_code,  sum(shipped_quantity)  AS total_items
FROM ACTIVITY_ITEM_TABLE
WHERE sl_type =  'salesorder'
       AND style_code IS  NOT  NULL
       AND shipped_quantity >  0
       AND in_period(sl_activity_ts, 'last1m')  =  TRUE
GROUP  BY style_code
ORDER  BY total_items DESC
LIMIT  10
```

Useful Helper Functions

1. in_period()

Checks if the given date is within the evaluated period based on the given expression.

Parameters

1. Date - the date in question
2. String - time period expression that will be converted to a start date and end date (see Time Period Expressions)

Return

- Return type is Boolean
- Returns true if the date in question is within the start and end date, false otherwise.

Usage

- in_period(<colum_with_timestamp>,'<period_expression>')

Example

```
SELECT  count(*)
FROM  ACTIVITY_TABLE
WHERE  in_period(sl_activity_ts,  'last1m')  =  TRUE
```

Notes

- When used in WHERE clause, please make sure to always provide a right-hand side value by evaluating the return with either TRUE or FALSE.

2. age()

Computes the age of the given date with respect to the current date.

Parameters

1. Date - the date whose age will be evaluated
2. String - Possible values: year, month, day. Defaults to year if given value is invalid

Return

- Return type is Integer
- Returns the age of the given date with respect to the specified time unit.

Usage

- age(<column_with_timestamp>,'<year|month|day>')

Example

```
SELECT  count(*)
FROM  MEMBER_TABLE
WHERE  age(member_since,  'month')  < 1
```

3. get_age_group()

Computes for the age group of the given date

Parameters

1. Date - the date whose age group (in years) will be evaluated

Return

- Return type is string
- Returns the age group of the given date, in years
- The age grouping is currently static:

- <15
- 15 - 24
- 25 - 34
- 35 - 44
- 45 - 54
- 55 - 64
- 65+
- Other

Usage

- get_age_group(<timestamp>)

Example

```
SELECT  get_age_group(birthdate)  AS  "Age Group",
       count(*)
FROM  MEMBER_TABLE
GROUP  BY  "Age Group"
```

4. Time Period Expressions

Period Type	Description	Example
MM/dd/yyyy-MM/dd/yyyy	Period between the first date and the second date, inclusive	'10/20/2018-11/30/2018'
MM/dd/yyyy	Period since first date	'12/20/2018'
alltime	all time	'alltime'
ytd	Year to date	'ytd'
mtd	Month to date	'mtd'
wtd	Week to date	'wtd'
last**<n>**d	last n days	'last5d'
last**<n>**w	last n weeks	'last3w'
last**<n>**m	last n months	'last2m'
prev**<n>**d	previous n days	'prev5d' *The 5 days before the last 5 days. Used for period-to-period comparison.*
prev**<n>**w	previous n weeks	'prev5w'

		The 5 weeks before the last 5 weeks. Used for period-to-period comparison.
prev**<n>**d	previous n months	'prev5' *The 5 months before the last 5 months. Used for period-to-period comparison.*

Note: All times are 12am midnight. Date is in UTC.

USE_SORT_MERGE_JOIN

Phoenix uses hash-joins, requiring the data to fit in memory. Problems may occur when a query JOINs one or more tables that have huge data sets (queries that JOIN very large tables such as ACTIVITY_TABLE and ACTIVITY_ITEM_TABLE are particularly susceptible to this). To avoid such problems, always use a query hint **USE_SORT_MERGE_JOIN**.

Sample Query

```
SELECT /*+ USE_SORT_MERGE_JOIN */
age_grp AS "Age Group",
    COUNT(DISTINCT member_id) AS "Member Count",
    COUNT(DISTINCT sl_member_id) AS "Active Members Count"
FROM(SELECT get_age_group(birthdate) as age_grp,
    member_id FROM MEMBER_TABLE) M
LEFT OUTER JOIN(SELECT sl_activity_ts,
    sl_member_id FROM ACTIVITY_TABLE WHERE in_period(sl_activity_ts, 'PERIOD')
= TRUE) A ON M.member_id = A.sl_member_id
GROUP BY "Age Group"
```

Double Quotes/Quoted Names

Double quotes are used for aliasing, usually for formatting the headers of the **SELECT** statement. They are case sensitive and can contain spaces.

Column Alias

```
SELECT member_id AS "Member Id",
    COALESCE(ACTIVITY_COUNT, 0) AS "Frequency"
FROM(SELECT SL_MEMBER_ID,
    COUNT(1) AS ACTIVITY_COUNT FROM ACTIVITY_TABLE AS ACTIVITIES WHERE
ACTIVITIES.SL_TYPE = 'sl_purchase'
    AND in_period(ACTIVITIES.SL_ACTIVITY_TS, 'last1y') = TRUE GROUP BY
SL_MEMBER_ID) AS A
RIGHT OUTER JOIN(SELECT member_id FROM MEMBER_TABLE) AS M
ON M.member_id = A.sl_member_id
```

Single Quotes

Used to reference string constants

```
SELECT 'New Member'
AS measure,
TO_DATE(TO_CHAR(member_since, 'yyyyMM'), 'yyyyMM') AS month,
    COUNT(1) AS val
FROM MEMBER_TABLE
WHERE in_period(member_since, 'alltime') = TRUE
GROUP BY month

SELECT * FROM
MEMBER_TABLE
WHERE first_name = 'John'
```

Unquoted names

Usually used for table names, column names, and aliasing. Unquoted names are not case sensitive. There is no maximum name length. On SELECT statements, it is possible to use unquoted names on each subquery then apply the formatting on the outermost SELECT statement.

```
SELECT member_id
FROM(SELECT member_id,
    COALESCE(ACTIVITY_COUNT, 0) AS frequency
    FROM(SELECT SL_MEMBER_ID,
        COUNT(1) AS ACTIVITY_COUNT
        FROM ACTIVITY_TABLE AS ACTIVITIES
        WHERE ACTIVITIES.SL_TYPE = 'sl_purchase'
        AND in_period(ACTIVITIES.SL_ACTIVITY_TS, 'last1y') = TRUE
        GROUP BY SL_MEMBER_ID) AS A
        RIGHT OUTER JOIN(SELECT member_id FROM MEMBER_TABLE) AS M
        ON M.member_id = A.sl_member_id)
WHERE frequency > 1
```

Phoenix Commands and Grammar

Please refer to Apache Phoenix Syntax documentation.

Native Phoenix Functions

Please refer to Apache Phoenix Native Function documentation.