



# 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**

**Copyright © 2021 Cheetah Digital, Inc.**

**All rights reserved.**

Printed in the United States of America

No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior agreement and written permission of Cheetah Digital, Inc.

Cheetah Digital, the Cheetah Digital logo, and other Cheetah names referenced herein are trademarks of Cheetah Digital, Inc., and may be registered in certain jurisdictions.

Other product names, designations, logos, and symbols may be trademarks or registered trademarks of their respective owners.

**PRODUCT MODULES AND OPTIONS.** This guide contains descriptions of modules that are optional and for which you may not have purchased a license. As a result, your software implementation may differ from descriptions in this guide. To find out more about the modules your organization has purchased, see your corporate purchasing agent or your Cheetah Digital sales representative.

**U.S. GOVERNMENT RESTRICTED RIGHTS.** Programs, Ancillary Programs and Documentation, delivered subject to the Department of Defense Federal Acquisition Regulation Supplement, are “commercial computer software” as set forth in DFARS 227.7202, Commercial Computer Software and Commercial Computer Software Documentation, and as such, any use, duplication and disclosure of the Programs, Ancillary Programs and Documentation shall be subject to the restrictions contained in the applicable Cheetah Digital license agreement. All other use, duplication and disclosure of the Programs, Ancillary Programs and Documentation by the U.S. Government shall be subject to the applicable Cheetah Digital license agreement and the restrictions contained in subsection (c) of FAR 52.227-19, Commercial Computer Software - Restricted Rights (June 1987), or FAR 52.227-14, Rights in Data—General, including Alternate III (June 1987), as applicable. Contractor/licensor is Cheetah Digital, Inc., 72 W Adams St 8th floor, Chicago, IL 60603.

### **PROPRIETARY INFORMATION NOTICE**

Cheetah Digital considers information included in this documentation and all Cheetah Digital documents to be Confidential Information. Your access to and use of this Confidential Information are subject to the terms and conditions of: (1) the applicable Cheetah Digital software license agreement, which has been executed and with which you agree to comply; and (2) the proprietary and restricted rights notices included in this documentation.



# Table of Contents

<b>Introduction</b>	<b>4</b>
<b>Creating Queries</b>	<b>4</b>
<b>Tables</b>	<b>5</b>
1. MEMBER_TABLE	5
Primary Key	5
Fields/Columns	5
Sample Queries	5
Basic	5
Aggregates and Grouping	5
2. ACTIVITY_TABLE	6
Primary Keys (Composite Keys)	6
Fields/Columns	6
Sample Queries	6
3. ACTIVITY_ITEM_TABLE	7
Primary Keys (Composite Keys)	7
Fields/Columns	7
Sample Queries	7
<b>Useful Helper Functions</b>	<b>8</b>
1. in_period()	8
Parameters	8
Return	8
Usage	8
Example	8
Notes	9
2. age()	9
Parameters	9
Return	9
Usage	9
Example	9
3. get_age_group()	9
Parameters	9
Return	9



Usage	10
Example	10
4. Time Period Expressions	10
<b>USE_SORT_MERGE_JOIN</b>	<b>11</b>
Sample Query	11
<b>Double Quotes/Quoted Names</b>	<b>11</b>
Column Alias	11
<b>Single Quotes</b>	<b>12</b>
<b>Unquoted names</b>	<b>12</b>
<b>Phoenix Commands and Grammar</b>	<b>12</b>
<b>Native Phoenix Functions</b>	<b>12</b>



## 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(<column_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' <i>The 5 days before the last 5 days. Used for period-to-period comparison.</i>
prev**<n>**w	previous n weeks	'prev5w'



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

*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, 'lastly') = 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, 'lastly') = 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](#).