Main Documentation

For the duplicates issue we had
Duplicates in the documentation file

📊 TABLE SELECTION PRIORITY (READ FIRST)

Notes

Always follow this hierarchy when writing queries:

🎯 QUERY DECISION MATRIX

Use this matrix to automatically select the correct table for user requests:

User Request Type	Use This Table	Why	Example
Daily metrics (revenue, DAU, purchases)	`agg_player_daily`	Pre-calculated, fast, cost-effective	"Revenue per day", "Daily active users"
Lifetime user metrics (LTV, total	`dim_player`	User-level lifetime attributes	"User LTV", "Total lifetime revenue",

User Request Type	Use This Table	Why	Example
revenue, install date)			"First-time depositors"
Chapter-specific daily analysis	`agg_player_chapter_daily`	Daily metrics broken down by chapter	"Revenue per chapter per day", "Users by chapter"
Event sequences, funnels, real-time debugging	`vmp_master_event_normalized`	Raw event detail required	"Event funnel analysis", "Current hour events"

Database Overview:

1. Platform: Google BigQuery
2. Primary Tables:
    - `yotam-395120.peerplay.agg_player_daily` (Most common - daily user metrics)
    - `yotam-395120.peerplay.dim_player` (User lifetime attributes)
    - `yotam-395120.peerplay.agg_player_chapter_daily` (Chapter-specific daily metrics)
    - `yotam-395120.peerplay.vmp_master_event_normalized` (Raw events - use only when aggregated tables don't have your data or when you need data from current date)

1. Main Table (Raw Events): `yotam-395120.peerplay.vmp_master_event_normalized`
    a. key columns in events table:
        i. date: our partition column, please use in every query that span over time to reduce costs
        - distinct_id: Unique identifier for each user
        - mp_event_name: Type of event (e.g., "purchase_successful", "merge", "generation")
        - JSON string representing the state of the game board
        - res_timestamp: Unix timestamp in milliseconds (stored as FLOAT64) for the event time (use this function to convert it to timestamp

TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)))

  - version_float: Version number of the application
  - chapter: Represents the user's progress level in the game
  - credit_balance: Tracks the user's virtual currency balance
  - item_id_1, item_id_2: Used in 'merge' events to identify merged items

Help with SQL. read about the database and then answer my question.

Events table Overview:

1. Platform: Google BigQuery
2. Main Table: `yotam-395120.peerplay.vmp_master_event_normalized`
   Key Columns:

- date: our partition column, please use in every query that span over time to reduce costs
- distinct_id: Unique identifier for each user
- mp_event_name: Type of event (e.g., "purchase_successful", "merge", "generation")
- JSON string representing the state of the game board
- res_timestamp: Unix timestamp in milliseconds (stored as FLOAT64) for the event time
- version_float: Version number of the application
- mp_os: Represents the operation system (Android or Apple)
- chapter: Represents the user's progress level in the game
- credit_balance: Tracks the user's virtual currency balance
- item_id_1, item_id_2: Used in 'merge' events to identify merged items
- **Important update** since version 0.368 in 2025-10-09 we started to have multiple different events with the same res_timestamp value. If the user needs to order the events by timestamp after this date, then do by: `res_timestamp, counter_per_session_game_side` (not just `res_timestamp` alone)

Additional notes when working with events table:

1. Don't try to partition floats - BQ doesn't allow that - you'll get: Partitioning by expressions of type FLOAT64 is not allowed
2. Event Tracking:
   - The game logs various user actions and game states as events.
   - Each event is associated with a specific user (distinct_id) and timestamp (res_timestamp).
   - Events can include purchases, game actions, board state changes, etc.
3. Time Representation:
   - Timestamps are stored in milliseconds since the Unix epoch as FLOAT64.
   - For date-based queries, use the date parameter
   - For liveops date needs, use res_liveops_date column
4. Game State:
   - The game_board column stores the game state as a JSON string.
   - It typically contains an array of items, some with suffixes like '.r' or '.l'.
   - Custom JavaScript functions within BigQuery are often used to parse and analyze this JSON data.
5. Version Control:
   - Events are associated with specific app versions (version_float).
   - Analysis often focuses on events from certain versions onwards (e.g., version 0.34 and above).
6. User Progress:
   - The 'chapter' column indicates the user's progress level in the game.
   - 'credit_balance' tracks the user's in-game virtual currency.
7. Event Types:
   - Various event types are tracked (mp_event_name), including:
     - 'purchase_successful' for completed purchases
     - 'merge' for combining game items

    - ‘generation’ for creating new items
    - Other game-specific actions
8. Complex Event Analysis:
    - Events are often analyzed in sequence or pairs to track changes over time.
    - Special attention is given to changes in the game board between events.
9. Data Filtering:
    - Queries often exclude null or empty game boards.
    - Specific event types may be filtered based on analysis requirements.
10. Custom Analysis Functions:
    - BigQuery allows for the creation of custom JavaScript functions (e.g., hasItem, itemDisappeared, calculateDelta).
11. ip - use client_ip

Important updates on Events table - make sure you read before compose the query

1. **Important update** since version 0.368 in 2025-10-09 we started to have multiple different events with the same res_timestamp value. If you need to order the after this date, then do by: `res_timestamp, counter_per_session_game_side`
2. Beware to filter correctly the fraudsters or potential fraudsters as in the fraudsters exclusion section - link

📋 AGGREGATED TABLE SCHEMAS

agg_player_daily

Purpose: Pre-calculated daily metrics per user (10-100x faster than raw events)
Available Metrics:

- **Revenue**: `total_ad_revenue`, `total_purchase_revenue`, `total_revenue`, `count_purchases`, `count_video_watched`
- **Ad Placements**: `total_ad_revenue_store`, `total_ad_revenue_bubble`, `count_video_watched_store`, `count_video_watched_bubble`
- **Game Actions**: `generations_count`, `generation_credits_spent`, `bubble_credits_spent`, `total_credits_spent`, `merges_count`
- **User Attributes**: `first_chapter`, `last_chapter`, `first_platform`, `last_platform`, `first_country`, `last_country`
- **Attribution**: `first_mediasource`, `last_mediasource`

dim_player

Purpose: User lifetime attributes and metrics
Available Metrics:

- **Lifetime Revenue**: `ltv_ad_revenue`, `ltv_iap_revenue`, `ltv_revenue`, `ltv_purchases`, `ltv_ads_impressions`
- **Lifetime Game Actions**: `ltv_generations_count`, `ltv_generation_credits_spent`, `ltv_bubble_credits_spent`, `ltv_total_credits_spent`, `ltv_merges_count`
- **Install Attributes**: `install_date`, `first_mediasource`, `first_country`, `first_platform`, `first_app_version`
- **Current State**: `last_chapter`, `last_credit_balance`, `last_platform`
- **Retention Metrics**: `d1_retention`, `d7_retention`, `d30_retention` (and other periods)
- **First Purchase**: `first_purchase_time`, `first_purchase_value`

agg_player_chapter_daily

Purpose: Same as `agg_player_daily` but with additional `chapter` sentrdimension for chapter-specific analysis

Important dimensions in the aggregated tables schema:

- first/last_platform (based on mp_os column from the raw events)
- first/last_country (based on mp_country_code column from the raw events)

- first/last_app_version (based on version_float from column the raw events)
- first/last_mediasource (based on mediasource column from the raw events)
- first/last_model (based on mp_model column from the raw events)

COMMON QUERIES WITH AGGREGATED TABLES

Always prefer aggregated table when the KPIs exist in those tables and current date data is not required:

Installs

```
SELECT  install_date AS date,
  COUNT(DISTINCT distinct_id) AS installs
FROM `yotam-395120.peerplay.dim_player`
    WHERE  install_date >= CURRENT_DATE() - 30  AND first_country NOT IN
('UA', 'IL','AM')
  AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.p
otential_fraudsters`)
GROUP BY install_date
ORDER BY install_date DESC
```

Daily Revenue

```
SELECT  date,
  SUM(total_purchase_revenue) AS purchase_revenue,
  SUM(total_ad_revenue) AS ad_revenue,
  SUM(total_revenue) AS total_revenue
FROM `yotam-395120.peerplay.agg_player_daily`
WHERE date >= CURRENT_DATE() - 30
GROUP BY date
ORDER BY date DESC
```

Daily Active Users (DAU)

```
SELECT  date,
  COUNT(DISTINCT apd.distinct_id) AS dau
FROM `yotam-395120.peerplay.agg_player_daily` apd
LEFT JOIN `yotam-395120.peerplay.dim_player` dp ON apd.distinct_id = dp.di
stinct_id
WHERE  date >= CURRENT_DATE() - 30  AND dp.first_country NOT IN ('UA', 'I
L','AM')
  AND apd.distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerp
lay.potential_fraudsters`)
GROUP BY date ORDER BY date DESC
```

ARPDAU (Average rate per DAU)

```
SELECT  date,
  COUNT(DISTINCT apd.distinct_id) AS dau,
  SUM(total_revenue) AS total_revenue,
  ROUND(SUM(total_revenue) / COUNT(DISTINCT apd.distinct_id), 4) AS arpd
au
FROM `yotam-395120.peerplay.agg_player_daily` apd
LEFT JOIN `yotam-395120.peerplay.dim_player` dp ON apd.distinct_id = dp.di
stinct_id
WHERE  date >= CURRENT_DATE() - 30  AND dp.first_country NOT IN ('UA', 'I
L','AM')
  AND apd.distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerp
lay.fraudsters`)
GROUP BY date
ORDER BY date DESC
```

Install version logic - using the raw events

1. Use it instead of dim_player only when you need also the current date data
2. Install Version Capture:

    - Identify the specific event that occurs when a user first installs the app ('impression_privacy').
    - This event should contain the version number of the app at the time of installation.

3. Logic Implementation:שרוק
    - Create a subquery or Common Table Expression (CTE) to isolate this information.
    - For each user, select the version associated with their install event.
    - Use DISTINCT to ensure one record per user, as there should only be one install event per user.
4. SQL Structure:

```
WITH install_versions AS (
  SELECT DISTINCT    distinct_id,
    version_float AS install_version
  FROM events_table
  WHERE event_name = 'impression_privacy')
```

5. Usage in Main Query:
    - Join this install version information with other event data as needed.
    - This allows associating all user actions with their original install version.
6. Key Considerations:
    - Ensure the install event is reliably tracked for all users.
    - Handle cases where a user might not have an install event (perhaps due to data loss or migration).
    - Be aware of any app updates that might occur very soon after installation, which could potentially mask the true install version.

This approach provides a foundation for analyzing user behavior based on the version they originally installed, allowing for cohort analysis and tracking how different app versions impact user engagement and retention over time.

Other KPIs using the aggregated tables:

Parameters and columns

Almost no parameter is a JSON so no need for logics like:

```
    COALESCE(SUM(CAST(JSON_EXTRACT_SCALAR(event_properties, '$.pric
e_usd') AS FLOAT64)), 0) AS total_purchase_usd
```

Instead, price_usd is just a regular column.

Event Funnel Analysis Best Practices

Avoid Assumptions About Event Sequence

When analyzing event funnels, it’s crucial to avoid making assumptions about the strict sequence of events unless explicitly specified. This is particularly important when dealing with time-based funnels.

Key Principle

Do not use LEAD() or LAG() functions to analyze funnels unless explicitly instructed to look at the immediately adjacent event.

Correct Approach

1. For time-based funnels, use a window of time to look for subsequent events, rather than assuming a strict sequence.
2. Join events based on user ID and timestamp conditions, allowing for flexibility in the event order within the specified timeframe.

Example

Instead of:

```
SELECT  *,
  LEAD(event_name) OVER (PARTITION BY user_id ORDER BY timestamp) AS
```

```
next_event
FROM events
```

Use:

```

```

When to Use LEAD() or LAG()

Only use these functions when:

1. You are explicitly told to analyze strictly adjacent events.
2. You are performing a step-by-step analysis of a user's journey where the exact sequence is crucial.
3. You are looking for immediate state changes or rapid successive actions.

Remember: Always clarify the exact requirements of the funnel analysis before making assumptions about event order or timing.

Game Modes Logic

Overview

The game has three different modes tracked through the `mode_status` column in the events table:

- Mode 1: `mode_status = 0`
- Mode 2: `mode_status = 1`
- Mode 3: `mode_status = 2`

Usage in Queries

When analyzing mode-specific behavior, especially for generation events, filter using the `mode_status` column. For example:

```
WHERE mp_event_name = 'generation'  AND mode_status = 2-- For Mode 3
```

Operating System Information

Overview

The operating system information is stored in the `mp_os` column in the `vmp_master_event_normalized` table. This is a string field that indicates the user's device operating system.

When you need lifetime data in which os user user first or last played use `first_platform` or `last_platform` columns from `dim_player`.

When you need daily data in which os user user first or last played use `first_platform` or `last_platform` columns from `agg_player_daily`.

Valid Values

The game currently supports only two operating systems:

- 'Apple': iOS devices
- 'Android': Android devices

1. Users cannot switch between iOS and Android platforms
2. Use ANY_VALUE() instead of MAX() or MIN() when getting a user's operating system, since users should only have one OS value

Chapter Progression Event

Event Name: 'scapes_tasks_new_chapter'
Description: This event tracks user progression through the game's chapter system. It is triggered whenever a user advances to a new chapter.

Key Characteristics:

- The 'chapter' parameter represents the new chapter the user is entering (not the completed one)
- To get the completed chapter number, subtract 1 from the 'chapter' value
- This event is triggered for every chapter progression, regardless of content completion status

- Multiple events per user are normal as they progress through chapters

Usage Example:

```
SELECT  distinct_id,
  CAST(chapter AS INT64) - 1 as completed_chapter,
  TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) as completion_time
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE mp_event_name = 'scapes_tasks_new_chapter' AND DATE >= CURRE
NT_DATE() - 7
```

Important: Exclude cases where CAST(chapter AS INT64) = 1, as this indicates the start of the first chapter rather than a completion.

To get per chapter the time the user played in it, total credits spent (generations and bubble), only generations and amount of generations

```
SELECT
  chapter,
  min(first_event_time)  first_event_time,
  max(last_event_time) last_event_time,
  sum(total_credits_spent) total_credits_spent,
  sum(generation_credits_spent) generation_credits_spent,
  sum(generations_count) generations_count
FROM `yotam-395120.peerplay.agg_player_chapter_daily`
WHERE distinct_id = '6809092b1b733bc32a4f4887' AND chapter<>0group b
y chapter
order by chapter
```

Example for users who purchase during chapter 1:

```
SELECT
  dp.distinct_id,
  dp.install_date,
  MIN(apcd.date) AS first_purchase_date,
  SUM(apcd.total_purchase_revenue) AS total_chapter1_purchase_revenue,
  SUM(apcd.count_purchases) AS total_chapter1_purchases
FROM `yotam-395120.peerplay.dim_player` dp
INNER JOIN `yotam-395120.peerplay.agg_player_chapter_daily` apcd
  ON dp.distinct_id = apcd.distinct_id
  AND apcd.chapter = 1
  AND apcd.total_purchase_revenue > 0
  AND apcd.date >= '2025-03-01'
WHERE
  dp.install_date >= '2025-03-01'
  AND dp.first_country NOT IN ('UA', 'IL','AM')
  AND dp.distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerpl
ay.fraudsters`)
GROUP BY
  dp.distinct_id,
  dp.install_date
ORDER BY
  dp.install_date DESC,
  first_purchase_date
```

Frenzy Events Query

Simple query to get Frenzy session results:

```
SELECT  -- Event details  distinct_id,  TIMESTAMP_MILLIS(CAST(res_timestam
p AS INT64)) AS event_time,
  mp_event_name,
  -- Frenzy identifiers  frenzy_id,  frenzy_event_id, --frenzy seasion_id  frenzy_
config_id, --the config id  live_ops_id, --the liveops id from the editor  -- Sessi
on metrics  is_jackpot, --indicator of the session ended with jackpot  rewards_
frenzy_total, --total credits from that session  ARRAY_LENGTH(JSON_EXTRAC
T_ARRAY(progress_count)) AS number_of_tries,  -- Performance data  explosi
on_count, --in that format ["try_1: 2","try_2: 22","try_3: 3"]  progress_count, --in
that format ["try_1: 2/15","try_2: 6/15","try_3: 3/15"]  -- Reward information  item
_id_1,  item_quantity_1,
  item_id_1_name,
  -- Standard fields  version_float,  chapter,
  credit_balance
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE mp_event_name IN ('frenzy_ended')
  AND date >= CURRENT_DATE - 1 ORDER BY distinct_id, res_timestamp, coun
ter_per_session_game_side
```

End of Content Parameter

The `end_of_content` parameter is a global parameter that indicates a user's content completion status. Once a user reaches the end of available content, this parameter will be set to 1 for ALL subsequent events until new content is released.

Key Characteristics:

- Boolean parameter (0 or 1)
- Present in every event type
- Acts as a persistent state flag:
  - When end_of_content = 0: User isn't in the end of the content
  - When end_of_content = 1: User has reached end of content and is currently in an EOC state
- The flag will remain 1 for all events (merges, generations, purchases, etc.) as long as the user is in EOC state
- Will reset to 0 when new content becomes available
- A user can go through multiple cycles of reaching EOC as new content is released

Usage Example:

```
-- Finding all actions a user took while in EOC stateSELECT  distinct_id,  mp_e
vent_name,
  dateFROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE end_of_content = 1and date >= current_date() - 7
```

End of Content in Chapter Progression

When ‘scapes_tasks_new_chapter’ events have end_of_content = 1, it represents a special case where the user has completed a chapter which is currently the last chapter in the available content.

Key Points:

- The same chapter can generate two ‘scapes_tasks_new_chapter’ events:
    1. First when entering the chapter (end_of_content = 0)
    2. Second when completing the chapter, which is currently the last chapter in the available content (end_of_content = 1)
- After this completion event, ALL subsequent events for this user will have end_of_content = 1 until new content is released
- A user can have multiple such completions over time as new content is released
- This combination provides the most accurate tracking of when users transition into EOC state

Usage Example:

```
-- Finding moments when users reached EOCSELECT  distinct_id,  CAST(chap
ter AS INT64) as completed_chapter,-- No subtraction needed when end_of_c
ontent = 1  TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) as eoc_time
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE mp_event_name = 'scapes_tasks_new_chapter'  AND end_of_content
= 1  AND DATE >= CURRENT_DATE - 1
```

Important Note: Unlike regular chapter progression events where we subtract 1 from the chapter value to get the completed chapter, when end_of_content = 1, the chapter value already represents the completed chapter since this event marks the actual completion of the final available chapter.

End of Content Events

Animations

All Stars Festival - animations

- FTUE
- event popup - opens / close / idle
- event popup - time's up
- how to popup
- points collected animation
- wining flows (Generic) .

Tournament Analytics

eoc_event_id, eoc_tournament_id, eoc_leaderboard_id, eoc_points_balance, eoc_user_rank ****will be super properties for all the events. The value is null if not set, never 0.

Each time a new event_id as a part of the tournament is open for users, a new event id will be assigned to all users.

board_task_new_task → goal_currency_id

Event Name	When the Event is Triggered	Additional Params	Param Logic
impression_eoc_popup	When the user sees the EOC pop-up		
eoc_event_started	When the user is assigned to a leaderboard	players_user_ids - a list of all user_ids of all bots (the bots user ids will be comma separated) eoc_end_time, timestamp	
click_eoc_lobby_badge	When the user clicks the EOC		

Event Name	When the Event is Triggered	Additional Params	Param Logic
	lobby badge		
impression_eoc_how_to_play	When the user sees the EOC how to play screen		
rewards_eoc	When the user gets a reward when the time's up		
click_eoc_go_board	When the user clicks the go button in the feature's board UI component		
impression_scapes	When the scapes screen is loaded	is_eoc_lobby_badge (boolean)	0 if there is no lobby badge, 1 if there is
ftue_flowX_stepY	When the user interact with the FTUE		

Audio

- Event-specific sound effects for chests, stars, and leaderboard interactions to enhance user experience are here ⏬⏬
- https://drive.google.com/open?id=1bNImIjxBhWrpGoVKS9AjAIh5QSWa7qzW&usp=drive_fs

Country Code Filtering Logic

In the data analysis queries, we apply a filtering condition based on country codes. Specifically, we exclude data from two countries: Ukraine ('UA'), Israel ('IL') and Armenia ('AM') and currency = 'UAH'

Implementation

The filtering is implemented using the following SQL condition:

```
AND (mp_country_code NOT IN ('UA', 'IL','AM') or currency != 'UAH')
```

This condition is applied in the initial data selection CTEs (Common Table Expressions) of our queries, typically in the `WHERE` clause along with other filtering conditions.

Same condition can be applied also in the aggregated tables using the following filter:

```
AND coalesce(first_country, last_country) NOT IN ('UA', 'IL','AM')
```

Exclude Low Payers Countries Users

In some cases we want to exclude/include users from low payers countries (this is list of countries we use for tests before publish the versions to our main selling countries.

To exclude users from low payers countries you can use this example:

```
 and mp_country_code not in
 (select country_code from yotam-395120.peerplay.dim_country where is_low
_payers_country=true)
```

Revenue Types and Sources

Total revenue is RV revenue (ads) + purchases

Rewarded Video (RV) Revenue - using the events

- Event Name: '`rewarded_video_revenue`' (below version 0.3652) and `rewarded_video_revenue_from_impression_data` (from version 0.3652 and above)
- Revenue Field: revenue

- Description: This event tracks revenue generated from rewarded video ad views
- Implementation: When calculating RV revenue, sum the `revenue` field for all `rewarded_video_revenue` and `rewarded_video_revenue_from_impression_data` events within the relevant time period

Example Query:

```
SELECT
  distinct_id,
  DATE(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) AS event_date,
  SUM(CAST(revenue AS FLOAT64)) AS rv_revenue
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE
  (
    -- For versions below 0.3652, use rewarded_video_revenue
    (CAST(version_float AS FLOAT64) < 0.3652 AND mp_event_name = 'rewarded_video_revenue')
    OR
    -- For versions 0.3652 and above, use rewarded_video_revenue_from_impression_data
    (CAST(version_float AS FLOAT64) >= 0.3652 AND mp_event_name = 'rewarded_video_revenue_from_impression_data')
  )
  AND date >= CURRENT_DATE() - 1
GROUP BY 1, 2
```

Rewarded Video (RV) KPIs - using dim_player

```
SELECT  distinct_id,
  ltv_ad_revenue AS total_ad_revenue,
  ltv_ads_impressions AS total_ads_watched,
  -- Calculate average ad revenue per impression if needed  CASE
    WHEN ltv_ads_impressions > 0
    THEN ROUND(ltv_ad_revenue / ltv_ads_impressions, 4)
```

```
   ELSE 0  END AS avg_revenue_per_ad
FROM `yotam-395120.peerplay.dim_player`
WHERE  -- Exclude test countries  first_country NOT IN ('UA', 'IL','AM')
  -- Exclude fraudsters  AND distinct_id NOT IN (SELECT distinct_id FROM `yot
am-395120.peerplay.fraudsters`)
  -- Only users who have watched at least one ad  AND ltv_ads_impressions >
0
ORDER BY ltv_ad_revenue DESC
```

Identifying Purchase Events

Purchase events are tracked in the `vmp_master_event_normalized` table with the event name 'purchase_successful'. When querying for purchase data, always filter for this specific event:

Exclusion of $0.01 Purchases

When analyzing purchase data, we exclude events with a price_original value of $0.01. This document explains the reasoning and implementation of this exclusion rule.

Apply this filter in all queries that analyze purchase behavior or revenue:

```
WHERE mp_event_name = 'purchase_successful'  AND CAST(price_original A
S FLOAT64) != 0.01
```

Exclusion of Fraudsters

We have two fraudsters related tables

- `yotam-395120.peerplay.fraudsters` - which contains **only** the revenue related fraudsters
- `yotam-395120.peerplay.potential_fraudsters` - which contains all fraudsters we found in the game.

When you need query that relates to revenue KPIs (use the mp_event_name='purchase_successful' then use `yotam-395120.peerplay.fraudsters` table, otherwise use `yotam-395120.peerplay.potential_fraudsters` table.

```
AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.po
tential_fraudsters`)
```

Or (depends on the case)

```
AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.po
tential_fraudsters`)
```

Unless specified otherwise, always exclude either fraudsters or potential_fraudsters (according the case as defined above) in the queries.

Example Implementation

```
SELECT  distinct_id,
  SUM(CAST(price_usd AS FLOAT64)) AS total_revenue
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE  mp_event_name = 'purchase_successful'  AND CAST(price_original A
S FLOAT64) != 0.01  AND distinct_id NOT IN (SELECT distinct_id FROM `yotam
-395120.peerplay.fraudsters`)
GROUP BY distinct_id
```

Purchase ID Deduplication Logic

Overview

When calculating purchase revenue, it’s critical to count each purchase exactly once by its unique `purchase_id` to avoid revenue inflation through duplicate event logging.

Implementation

The recommended approach for deduplicating purchases uses a two-step process:

1. Group data by both `custom_day` and `purchase_id`
2. Take the maximum value of `price_usd` for each unique combination

SQL Pattern

```
sql
WITH deduplicated_purchases AS (
  SELECT-- Define your time period (e.g., custom day starting at 10am)    DATE
(TIMESTAMP_ADD(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)),
        INTERVAL -10 HOUR)) AS custom_day,
   purchase_id,
    -- Take the max price in case the same purchase_id appears multiple times
MAX(CAST(price_usd AS FLOAT64)) AS purchase_amount
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE mp_event_name = 'purchase_successful'    -- Apply standard purcha
se filters    AND CAST(price_original AS FLOAT64) != 0.01    AND distinct_id N
OT IN (SELECT distinct_id FROM `yotam-395120.peerplay.fraudsters`)
    AND mp_country_code NOT IN ('UA', 'IL', 'AM')
    AND COALESCE(currency, '') != 'UAH'  -- Exclude UAH currency purchases
AND DATE >= CURRENT_DATE - 1  GROUP BY custom_day, purchase_id
)
-- Further aggregation can now be performed on deduplicated dataSELECT  c
ustom_day,
  SUM(purchase_amount) AS total_purchase_revenue
FROM  deduplicated_purchases
GROUP BY  custom_day'AM'
```

Generator and Item Tracking

The generator_id parameter indicates which generator created the items in a generation event. A generator itself is an item with additional generation capabilities - it has the same item_id as its generator_id.

When analyzing generation events:

- For item analysis: Focus on the item_ids in the generated_items parameter
- For generator analysis: Use the generator_id parameter to identify which generator produced the items

Generator Items

To see what items a specific user generated:

```
SELECT  -- Event details  distinct_id,
  TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS event_time,
  mp_event_name AS event_name,
  version_float, --app version  mp_os,
  number_of_events, --num of generations in this event (in case of batching)
  generator_name,
  generator_capacity_left,
  item_id_1_chain,
  item_id_1_level,
  item_id_1_name,
  delta_credits, --always negative, amount of credits substructed from balance
credit_balance,
  metapoint_balance,
  chapter,
  generated_items
  FROM  `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE mp_event_name in ('generation')
  AND DATE = CURRENT_DATE()
  AND distinct_id='6809092b1b733bc32a4f4887' --change to your id  order b
y event_time desc-- Limit to a reasonable number of rows for initial exploratio
nLIMIT 100;
```

The `is_unified` parameter indictes if the `generatation` events contains multiple generation actions or only single one.

In case `is_unified=1`, then `generated_items` will contain description of all generation like here:

```
[{"index":0,"item_chain":601,"item_id":610,"item_level":4,"item_name":"Club S
andwich"},
{"index":1,"item_chain":601,"item_id":609,"item_level":3,"item_name":"PB\u00
26J"},
{"index":2,"item_chain":1800,"item_id":1802,"item_level":3,"item_name":"Tea B
owl"}]
```

In case `is_unified<>1`, then those parameters will contain the generation details:

```
 item_id_1,
 item_id_1_name,
 item_id_1_chain,
 item_id_1_level,
 item_id_1_level
```

This query gives the amount of times each item was generated yesterday:

```
WITH generation_details AS (
  SELECT
    -- For unified generations (batch), extract items from JSON array
    CASE
      WHEN is_unified = 1 THEN
        JSON_EXTRACT_ARRAY(generated_items)
      ELSE NULL
    END AS generated_items_array,
    -- For single generations, use item_id_1_name directly
    CASE
      WHEN is_unified <> 1 THEN item_id_1_name
      ELSE NULL
    END AS single_generated_item,
    is_unified
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE
    mp_event_name = 'generation'
    AND date = CURRENT_DATE() - 1  -- Yesterday only
    AND mp_country_code NOT IN ('UA', 'IL','AM')
    AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.
potential_fraudsters`)
),

all_generated_items AS (
  -- Extract item_name from unified generations
  SELECT
    JSON_EXTRACT_SCALAR(generated_item, '$.item_name') AS item_name
  FROM generation_details,
  UNNEST(generated_items_array) AS generated_item
  WHERE is_unified = 1
    AND JSON_EXTRACT_SCALAR(generated_item, '$.item_name') IS NOT NUL
L
    AND JSON_EXTRACT_SCALAR(generated_item, '$.item_name') != ''

  UNION ALL

  -- Get item_name from single generations
  SELECT single_generated_item AS item_name
  FROM generation_details
  WHERE is_unified <> 1
    AND single_generated_item IS NOT NULL
    AND single_generated_item != ''
)

SELECT
  item_name,
  COUNT(*) AS total_generations
FROM all_generated_items
```

```
GROUP BY item_name
ORDER BY total_generations DESC
```

This query return how much times "Tea Bowl" item was generated in the last 7 days per day:

```
WITH generation_details AS (
  SELECT
    date,
    -- For unified generations (batch), count Tea Bowl occurrences in JSON arr
ay
    CASE
      WHEN is_unified = 1 THEN (
        SELECT COUNT(*)
        FROM UNNEST(JSON_EXTRACT_ARRAY(generated_items)) AS generate
d_item
        WHERE JSON_EXTRACT_SCALAR(generated_item, '$.item_name') = 'Tea
Bowl'
      )
      -- For single generations, check if item is Tea Bowl
      WHEN is_unified <> 1 THEN (
        CASE
          WHEN item_id_1_name = 'Tea Bowl' THEN 1
          ELSE 0
        END
      )
      ELSE 0
    END AS tea_bowl_generations
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE
    mp_event_name = 'generation'
    AND date >= CURRENT_DATE() - 7  -- Last 7 days
    AND mp_country_code NOT IN ('UA', 'IL', 'AM')
    AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.
potential_fraudsters`)
)

SELECT
  date,
  SUM(tea_bowl_generations) AS total_tea_bowl_generations
FROM generation_details
GROUP BY date
--HAVING SUM(tea_bowl_generations) > 0
ORDER BY date DESC
```

Merged Items

The `merge` event indicates on merges which has been made by the user.

To get which merges were made by the user:

```
SELECT
    -- Event details
    distinct_id,
    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS event_time,
    mp_event_name AS event_name,
    version_float, -- app version
    mp_os,
    number_of_events, -- num of generations in this event (in case of batching)
    credit_balance,
    metapoint_balance,
    delta_credits, -- expected to be null/0 in merges
    chapter,
    theoretical_chapter, -- chapter is user was using all metapoints
    game_board,
    is_joker, -- indication if merge was made with joker
    is_semi_locked,
    is_unified,
    merged_items,
    counter_per_session_game_side
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE
    mp_event_name IN ('merge')
    AND DATE >= CURRENT_DATE() - 1
    AND distinct_id = '6809092b1b733bc32a4f4887'
ORDER BY event_time, counter_per_session_game_side DESC
-- Limit to a reasonable number of rows for initial exploration
LIMIT 100;
```

If is_unified=1 then the merge event describe batch of merges made by the player.

The merges that made by the user are described in merged_item parameter, which is json with element per each merge.

Example:(user made 3 merges in this is what we got in merged_items

```
[{"index":0,"item_id_1":7041,"item_id_1_chain":7040,"item_id_1_level":2,"item_id_1_name":"Action Slate","item_id_2":null,"item_id_2_chain":null,"item_id_2_level":null,"item_id_2_name":null},
{"index":1,"item_id_1":7045,"item_id_1_chain":7040,"item_id_1_level":6,"item_id_1_name":"Movie Recorder","item_id_2":null,"item_id_2_chain":null,"item_id_2_level":null,"item_id_2_name":null},
{"index":2,"item_id_1":7046,"item_id_1_chain":7040,"item_id_1_level":7,"item_id_1_name":"VIP Carpet","item_id_2":null,"item_id_2_chain":null,"item_id_2_level":null,"item_id_2_name":null}]
```

If is_unified<>1 then the merge event describe a single merge action.

The merged items are described in the following parameters:

```
    item_id_1,
    item_id_1_chain,
    item_id_1_level,
    item_id_1_name,
    item_id_2,
    item_id_2_chain,
    item_id_2_level,
    item_id_2_name
```

Query to get the amount of merges user made per day with “small mixer” item:

```
WITH merge_details AS (
  SELECT
    date,
    CASE
      -- For unified merges (batch), parse the JSON merged_items
      WHEN is_unified = 1 THEN (
        SELECT COUNT(*)
        FROM UNNEST(JSON_EXTRACT_ARRAY(merged_items)) AS merge_item
        WHERE JSON_EXTRACT_SCALAR(merge_item, '$.item_id_1_name') = 'Sm
all Mixer'
           OR JSON_EXTRACT_SCALAR(merge_item, '$.item_id_2_name') = 'Small
Mixer'
      )
      -- For single merges, check the direct parameters
      WHEN is_unified <> 1 THEN (
        CASE
          WHEN item_id_1_name = 'Small Mixer' OR item_id_2_name = 'Small Mixe
r' THEN 1
          ELSE 0
        END
      )
      ELSE 0
    END AS merges_with_small_mixer
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE
    mp_event_name = 'merge'
    AND date >= CURRENT_DATE() - 5  -- Last 5 days
    AND mp_country_code NOT IN ('UA', 'IL','AM')
    AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.
fraudsters`)
)

SELECT
  date,
  SUM(merges_with_small_mixer) AS total_small_mixer_merges
FROM merge_details
GROUP BY date
ORDER BY date DESC
```

Query to get the how much each item was merged yesterday:

```
WITH merge_details AS (
  SELECT
    -- For unified merges (batch), extract item_id_1_name from each merge in th
e JSON array
    CASE
      WHEN is_unified = 1 THEN
        JSON_EXTRACT_ARRAY(merged_items)
      ELSE NULL
    END AS merged_items_array,
    CASE
      WHEN is_unified <> 1 THEN item_id_1_name
      ELSE NULL
    END AS single_merge_item,
    is_unified
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE
    mp_event_name = 'merge'
    AND date = CURRENT_DATE() - 1  -- Yesterday only
    AND mp_country_code NOT IN ('UA', 'IL','AM')
    AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.
fraudsters`)
),

all_merge_items AS (
  -- Extract item_id_1_name from unified merges
  SELECT
    JSON_EXTRACT_SCALAR(merge_item, '$.item_id_1_name') AS item_name
  FROM merge_details,
  UNNEST(merged_items_array) AS merge_item
  WHERE is_unified = 1
    AND JSON_EXTRACT_SCALAR(merge_item, '$.item_id_1_name') IS NOT NU
LL
    AND JSON_EXTRACT_SCALAR(merge_item, '$.item_id_1_name') != ''

  UNION ALL

  -- Get item_id_1_name from single merges
  SELECT single_merge_item AS item_name
  FROM merge_details
  WHERE is_unified <> 1
    AND single_merge_item IS NOT NULL
    AND single_merge_item != ''
)

SELECT
  item_name,
  COUNT(*) AS total_merges
FROM all_merge_items
GROUP BY item_name
ORDER BY total_merges DESC
```

game_board parameter

The game_board parameter is a JSON array string representing the state of each tile on the game board. Each element in the array follows the format "ITEM_ID.SUFFIX" where:

1. Format Details:
    - ITEM_ID: A numeric identifier for an item (e.g., 200, 508, 903). When the tile is empty, ITEM_ID is always 0
    - Period (.) separator
    - SUFFIX: A single letter indicating the state of the item or tile:
        - 'r': regular - item can be used and merged normally
        - 'l': locked - item cannot be used or merged
        - 's': semi-locked - item cannot be used but can be merged
        - 'e': empty tile - only used with ITEM_ID 0, representing no item on this tile
        - 'b': bubble - item is inside a bubble and needs to be popped before it can be used otherwise it will be removed.
2. Example of a game_board value:

```
["450.r", "512.l", "680.s", "0.e", "720.r", "0.e", "512.b"]
```

1. Key characteristics:
    - Each element is enclosed in quotes
    - Elements are comma-separated
    - The entire array is enclosed in square brackets
    - No duplicate ITEM_ID.SUFFIX combinations should exist in a valid game board
    - Empty tiles are always represented as "0.e"

Analyzing Game Boards

1. Getting Latest Game Board:

```

WITH latest_boards AS (
  SELECT
    distinct_id,
    game_board,
    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) as event_time,
    counter_per_session_game_side
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE game_board IS NOT NULL
  AND DATE >= CURRENT_DATE - 14
  QUALIFY ROW_NUMBER() OVER(PARTITION BY distinct_id ORDER BY res_ti
mestamp,counter_per_session_game_side DESC) = 1
)
```

Checking for Items:

```
-- Standard check for regular items only (default approach)
REGEXP_CONTAINS(game_board, r'"123\\.r"')

-- Special cases (only when specifically needed):
-- Check for any state (regular, locked, semi-locked, or bubble)
REGEXP_CONTAINS(game_board, r'"123\\.[rlsb]"')
-- Check for locked items only
REGEXP_CONTAINS(game_board, r'"123\\.l"')
-- Check for semi-locked items only
REGEXP_CONTAINS(game_board, r'"123\\.s"')
-- Check for bubble items only
REGEXP_CONTAINS(game_board, r'"123\\.b"')
```

1. Common Analysis Patterns:

```
-- Binary flags for multiple items (default - regular items only)
SELECT
  distinct_id,
  CASE WHEN REGEXP_CONTAINS(game_board, r'"211\\.r"') THEN 1 ELSE 0 E
ND as has_211,
  CASE WHEN REGEXP_CONTAINS(game_board, r'"212\\.r"') THEN 1 ELSE 0 E
```

```
ND as has_212
FROM latest_boards

-- Counting users with item combinations
SELECT
  COUNT(DISTINCT distinct_id) as total_users,
  COUNTIF(REGEXP_CONTAINS(game_board, r'"211\\.r"') AND
          REGEXP_CONTAINS(game_board, r'"212\\.r"')) as users_with_both_item
s
FROM latest_boards
```

1. Parsing approaches:
   - For counting suffixes: Use string operations like LENGTH() and REPLACE()

     ```
     sql
     Copy(LENGTH(game_board) - LENGTH(REPLACE(game_board, '.r',
     ''))) / LENGTH('.r')
     ```

   - For analyzing individual items: Parse as JSON array and process each item
2. Common validation needs:
   - Check for duplicate item IDs
   - Count occurrences of different states (regular, locked, semi-locked, empty)
   - Validate array structure and format
   - Verify that empty tiles are correctly formatted as “0.e”

Example of default vs special case analysis:

```
sql
-- Standard item check (default behavior - only regular items)SELECT  distinct
_id,
  CASE WHEN REGEXP_CONTAINS(game_board, r'"211\\.r"') THEN 1 ELSE 0 E
ND as has_211
FROM latest_boards
```

```
-- Special case when all states need to be analyzed (must be explicitly reques
ted)SELECT  distinct_id,
  CASE WHEN REGEXP_CONTAINS(game_board, r'"211\\.[rlsb]"') THEN 1 ELSE
0 END as has_211_any_state
FROM latest_boards
```

Board Task Values

Overview

In case an analyst would like to know what is the real value it will cost the user to finish a board task, then we have this table for it in the database `yotam-395120.peerplay.board_tasks_values` (the table is partitioned by date).
The table contains the board task items (up to 3 items), the required credits value to get each of them, and the value to complete the entire task (notion for the query that builds this table - link).

Examples

Get avg credits required for task per chapter in specific version

```
SELECT chapter, version, avg(task_value) as avg_algo_task_value
FROM `peerplay.board_tasks_values`
where task_value<>0
and board_task_trigger='Algo'
and date>=current_date-30
and version=0.367
group by 1,2
order by 1,2
```

Tracking Low Balance Cases

Overview

To accurately track low balance cases, we need to identify unique instances where a user's balance drops below a threshold. A "case" starts when the balance drops below the threshold and ends when it goes back above it. Multiple events with low balance in sequence count as a single case.

Implementation

```
WITH balance_changes AS (
  SELECT    distinct_id,
    res_timestamp,
    counter_per_session_game_side,
    credit_balance,
-- Track when balance transitions from high to low    LAG(CASE WHEN credit_
balance >= threshold THEN 1 ELSE 0 END) OVER(
      PARTITION BY distinct_id
      ORDER BY res_timestamp, counter_per_session_game_side
    ) as prev_was_high,
    CASE WHEN credit_balance < threshold THEN 1 ELSE 0 END as is_low
  FROM events_table
  WHERE credit_balance IS NOT NULL),
low_balance_cases AS (
  SELECT    distinct_id,
-- Count only transitions from high to low    COUNT(CASE      WHEN prev_was
_high = 1 AND is_low = 1      THEN 1    END) as low_balance_cases
  FROM balance_changes
  GROUP BY distinct_id
)counter_per_session_game_side
```

Key Components

1. **Balance State Tracking**:
   - Define high/low states using a threshold
   - Use `LAG` window function to look at previous balance state

    - Create boolean flags for state tracking:
        - `prev_was_high`: Was the previous balance above threshold?
        - `is_low`: Is the current balance below threshold?

2. **Case Counting Logic**:
    - A new case is counted ONLY when:
        - Previous balance was high (`prev_was_high = 1`)
        - Current balance is low (`is_low = 1`)
    - This ensures we only count the transition points

Example Scenario

Given a threshold of 3 credits:

```
Copy
Event Sequence:
1. Balance: 5  (High)
2. Balance: 2  (Low)  → CASE #1 STARTS
3. Balance: 1  (Low)  → Same case
4. Balance: 0  (Low)  → Same case
5. Balance: 4  (High) → Case #1 ends
6. Balance: 2  (Low)  → CASE #2 STARTS
7. Balance: 5  (High) → Case #2 ends
```

In this sequence, despite having 4 events with low balance, we count only 2 cases because there were only 2 transitions from high to low balance.

Usage Notes

- Always filter out NULL balance values
- Order by `res_timestamp` and `counter_per_session_game_side` to ensure correct sequence analysis
- Consider additional filters (country codes, date ranges, etc.) based on analysis needs

- The logic can be adapted for different thresholds or balance conditions

This pattern can be used whenever you need to track distinct occurrences of any state change in your data, not just balance changes.

Task Value and Reward Calculation Logic

Task Value Mapping

For both events, goal items are mapped using:

- Table: `yotam-395120.peerplay.item_mapping_to_value` as mapping_table
- Join condition: `CAST(mapping_table.distinct_id AS INT64) = event.goal_item_id_N`
- Value field: `real value` column from mapping table
- Mode adjustment: When mode_status = 2, multiply the mapped value by 0.75

1. 'rewards_board_task' Event

This event represents when a task is completed and rewards are given.

Value Calculation

```
-- Join for each goal itemLEFT JOIN `yotam-395120.peerplay.item_mapping_t
o_value` m1
  ON CAST(m1.distinct_id AS INT64) = e.goal_item_id_1
LEFT JOIN `yotam-395120.peerplay.item_mapping_to_value` m2
  ON CAST(m2.distinct_id AS INT64) = e.goal_item_id_2
LEFT JOIN `yotam-395120.peerplay.item_mapping_to_value` m3
  ON CAST(m3.distinct_id AS INT64) = e.goal_item_id_3
-- Value calculation for each goalCASE  WHEN goal_item_id_N IS NOT NULL T
HEN    CASE      WHEN mode_status = 2      THEN SAFE_CAST(SAFE_CAST(CO
ALESCE(mN.`real value`, '0') AS FLOAT64) * 0.75 AS INT64)
      ELSE SAFE_CAST(COALESCE(mN.`real value`, '0') AS INT64)
    END  ELSE 0END
```

Reward Calculation

```
CASE  WHEN (item_id_1 = 1 OR item_id_2 = 1 OR item_id_3 = 1)
  THEN COALESCE(CAST(CASE WHEN item_id_1 = 1 THEN item_quantity_1 ELS
E 0 END AS INT64), 0) +       COALESCE(CAST(CASE WHEN item_id_2 = 1 THE
N item_quantity_2 ELSE 0 END AS INT64), 0) +       COALESCE(CAST(CASE W
HEN item_id_3 = 1 THEN item_quantity_3 ELSE 0 END AS INT64), 0)
  ELSE 0END
```

2. 'board_tasks_new_task' Event

This event represents when a new task is created/assigned.

Value Calculation

Identical table and mapping logic as rewards_board_task event.

Reward Calculation

```
(
  CASE    WHEN goal_currency_id_1 = 'Credits'    THEN COALESCE(CAST(goal_
currency_quantity_1 AS INT64), 0)
    ELSE 0  END +  CASE    WHEN goal_currency_id_2 = 'Credits'    THEN COAL
ESCE(CAST(goal_currency_quantity_2 AS INT64), 0)
    ELSE 0  END)
```

Key Differences Between Events

1. Event Timing
    - rewards_board_task: Triggered at task completion
    - board_tasks_new_task: Triggered at task creation
2. Reward Structure
    - rewards_board_task: Uses item_id/item_quantity fields, where credits are identified by item_id = 1

  - board_tasks_new_task: Uses goal_currency_id/goal_currency_quantity fields, where credits are identified by goal_currency_id = “Credits”

I’ll help revise that paragraph to make the version changes and delta_credits behavior clearer. Here’s how it should read:

Generation Events Logic

Starting from version 0.357, generation events were optimized to batch multiple generations into a single event (but it’s not in all events, therefore use the column is_unified to know if generation event contains batch of generations or not). The number_of_events parameter indicates how many generations occurred in the batch. However, the way delta_credits was recorded changed across versions:

- In versions before 0.357: Events were not batched, and delta_credits represented the cost of a single generation
- In versions 0.357 to 0.35923: Events were batched, but delta_credits still represented the cost of just one generation. Therefore, we needed to multiply by number_of_events to get the total cost
- From version 0.35931 onwards: Events are batched, and delta_credits was fixed to represent the total cost for all generations in the batch

Credit Cost Calculation:

- For versions 0.357 to 0.35923: Total cost = delta_credits * number_of_events (because delta_credits only represented one generation’s cost)
- For all other versions: Total cost = delta_credits (because delta_credits already represents the total cost)

Implementation Example:

```
SUM(
  CASE
    WHEN mp_event_name = 'generation'
      AND CAST(version_float AS FLOAT64) >= 0.357
      AND CAST(version_float AS FLOAT64) <= 0.35923
      AND number_of_events IS NOT NULL
    THEN CAST(delta_credits AS INT64) * CAST(number_of_events AS INT64)
    ELSE CAST(delta_credits AS INT64)
  END
) * -1 AS total_credit_spend
```

Would you like me to update the rest of the documentation to reflect this clearer explanation of how delta_credits behavior changed across versions?

Segment Parsing Documentation for Game Analytics

This document explains how to parse both `server_segments` and `firebase_segments` fields in BigQuery to extract meaningful data for analysis.

Understanding Segment Formats

Server Segments

The `server_segments` field contains data in the following format:

```
["KeyName1;value1.value2.value3","KeyName2;singleValue","KeyName3;comp1.comp2"]
```

Key characteristics:

- Stored as a JSON array of strings
- Each string follows the pattern `KeyName;ValueString`
- Values may contain multiple components separated by periods (`.`)
- Common keys include: `LiveOpsData`, `ItemData`, `FlowersFeatureConfigData`, `FusionFairFeatureConfigData`, `ScapeGoalData`

Example:

```
["LiveOpsData;timed_task_11.rare_chain_1.timed_task_1","FusionFairFeatureConfigData;fusion_1","ItemData;rare_chain_1","FlowersFeatureConfigData;flowers_1"]
```

Firebase Segments

The `firebase_segments` field follows this format:

```
["KeyName1.value1","KeyName2.value2","KeyName3.{\\\\"json\\\\":\\\\"value
\\\\"}"]
```

Key characteristics:

- Stored as a JSON array of strings
- Each string follows the pattern `KeyName.ValueString`
- Values are typically single items, but may sometimes contain JSON objects
- Common keys include: `BoardData`, `ItemData`, `LiveOpsData`, `ChapterConfigData`, etc.

Example:

```
["BoardData.default","ItemData.rare_chain_f","LiveOpsData.sale","SmartlookC
onfigData.{\\\\"EnableSmartlook\\\\":true,\\\\"RecordingSampleRate\\\\":1,\\\\"Ma
xSessionCount\\\\":0}"]
```

Parsing Techniques in BigQuery

1. Extracting Full Values for a Given Key

To extract the complete value after the separator for a specific key:

For Server Segments (semicolon separator):

```
-- Extract the full LiveOpsData valueCASE WHEN REGEXP_CONTAINS(server_
segments, r'LiveOpsData;([^"]+)')
    THEN REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)')
    ELSE NULL END AS server_LiveOpsData_full
```

For Firebase Segments (dot separator):

```
-- Extract the full ItemData valueCASE WHEN REGEXP_CONTAINS(firebase_se
gments, r'"ItemData\\\\.([^"]+)"')
    THEN REGEXP_EXTRACT(firebase_segments, r'"ItemData\\\\.([^"]+)"')
    ELSE NULL END AS firebase_ItemData
```

2. Breaking Down Dot-Separated Components (Server Segments)

When server segment values contain multiple dot-separated components:

```
-- Extract first component of LiveOpsDataCASE WHEN REGEXP_CONTAINS(s
erver_segments, r'LiveOpsData;([^"]+)')
    THEN SPLIT(REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)'),
'.')[SAFE_OFFSET(0)]
    ELSE NULL END AS server_LiveOpsData_comp1,
-- Extract second component of LiveOpsDataCASE WHEN REGEXP_CONTAIN
S(server_segments, r'LiveOpsData;([^"]+)')
    THEN SPLIT(REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)'),
'.')[SAFE_OFFSET(1)]
    ELSE NULL END AS server_LiveOpsData_comp2
```

The `SAFE_OFFSET` function is crucial here as it returns NULL if the component doesn’t exist, preventing errors.

Examples to get shop segment and po segment:

```
   COALESCE(
     CASE WHEN REGEXP_CONTAINS(server_segments, r'StoreContainerData;
([^"]+)')
         THEN REGEXP_EXTRACT(server_segments, r'StoreContainerData;([^"]
+)')
         ELSE NULL END,
     'default'
   ) AS shop_segment,
      COALESCE(
     CASE WHEN REGEXP_CONTAINS(server_segments, r'PersonalOfferConfig
Data;([^"]+)')
         THEN REGEXP_EXTRACT(server_segments, r'PersonalOfferConfigData;
([^"]+)')
         ELSE NULL END,
     'default'
   ) AS po_segment,
```

3. Handling Variable Numbers of Components

Some segment values may have a variable number of dot-separated components. Use multiple columns with `SAFE_OFFSET` to accommodate this variability:

```
CASE WHEN REGEXP_CONTAINS(server_segments, r'LiveOpsData;([^"]+)')
    THEN SPLIT(REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)'),
'.')[SAFE_OFFSET(0)]
    ELSE NULL END AS comp1,
CASE WHEN REGEXP_CONTAINS(server_segments, r'LiveOpsData;([^"]+)')
    THEN SPLIT(REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)'),
'.')[SAFE_OFFSET(1)]
    ELSE NULL END AS comp2,
-- Add as many components as needed
```

4. Complete Example for Server Segments Processing

```
SELECT  -- Extract full values for each key  CASE WHEN REGEXP_CONTAINS
(server_segments, r'LiveOpsData;([^"]+)')
      THEN REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)')
      ELSE NULL END AS server_LiveOpsData_full,
  CASE WHEN REGEXP_CONTAINS(server_segments, r'ItemData;([^"]+)')
      THEN REGEXP_EXTRACT(server_segments, r'ItemData;([^"]+)')
      ELSE NULL END AS server_ItemData_full,
  -- Extract components for LiveOpsData  CASE WHEN REGEXP_CONTAINS(se
rver_segments, r'LiveOpsData;([^"]+)')
      THEN SPLIT(REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]
+)'), '.')[SAFE_OFFSET(0)]
      ELSE NULL END AS server_LiveOpsData_comp1,
  CASE WHEN REGEXP_CONTAINS(server_segments, r'LiveOpsData;([^"]+)')
      THEN SPLIT(REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]
+)'), '.')[SAFE_OFFSET(1)]
      ELSE NULL END AS server_LiveOpsData_comp2,
  CASE WHEN REGEXP_CONTAINS(server_segments, r'LiveOpsData;([^"]+)')
      THEN SPLIT(REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]
+)'), '.')[SAFE_OFFSET(2)]
      ELSE NULL END AS server_LiveOpsData_comp3
FROM your_table
```

Best Practices

1. **Always use SAFE_OFFSET()** when accessing array elements to avoid errors with component indices that don't exist.
2. **Extract both full values and components** to allow for both high-level and detailed analyses.
3. **Use REGEXP_CONTAINS before REGEXP_EXTRACT** to prevent trying to extract from null or non-matching segments.
4. **Create clear column naming conventions** that distinguish:
    - The segment source (server vs. firebase)
    - The key name (LiveOpsData, ItemData, etc.)
    - Whether it's a full value or component number
5. **Handle missing segments gracefully** by using CASE WHEN and providing NULL values when segments don't exist.

Performance Considerations

1. **Avoid repeatedly parsing the same segment keys** in multiple places in your query. Use Common Table Expressions (CTEs) to parse the segments once and reuse the results.

2. **Consider materializing frequently analyzed segment data** into a separate table or view to avoid repeated parsing.
3. **Limit the number of components you extract** to those actually needed for analysis to reduce query complexity.

Example of Comparing Segments Between Events

This pattern is particularly useful when comparing segments between different events for the same user:

```
WITH event1 AS (
  SELECT    distinct_id,
    CASE WHEN REGEXP_CONTAINS(server_segments, r'LiveOpsData;([^"]+)')
         THEN REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)')
         ELSE NULL END AS LiveOpsData
  FROM events  WHERE event_type = 'type1'),
event2 AS (
  SELECT    distinct_id,
    CASE WHEN REGEXP_CONTAINS(server_segments, r'LiveOpsData;([^"]+)')
         THEN REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)')
         ELSE NULL END AS LiveOpsData
  FROM events  WHERE event_type = 'type2')
SELECT  e1.distinct_id,
  e1.LiveOpsData AS event1_LiveOpsData,
  e2.LiveOpsData AS event2_LiveOpsData,
  CASE WHEN e1.LiveOpsData = e2.LiveOpsData THEN 'Same' ELSE 'Differen
t' END AS comparison
FROM event1 e1
JOIN event2 e2 ON e1.distinct_id = e2.distinct_id
```

This allows you to easily identify segment changes between events, which can be crucial for troubleshooting issues like authentication errors or unexpected behavior changes.

Summing up rewards and credits spend

Credits Spend Logic

Generation Spend

Event: `generation`

Column: `delta_credits`

Important Note: `delta_credits` is a non-positive integer

Calculation:

`SUM(ABS(CAST(delta_credits AS INT64)))`

When calculating credit spend from generations:

- Use the absolute value of `delta_credits` since it's non-positive
- The sum of absolute values = total credits spent on generations
- A larger absolute value = more credits spent
- Example: `delta_credits = -30` → user spent 30 credits

Example query:

```
SELECT
  distinct_id,
  SUM(ABS(CAST(delta_credits AS INT64))) AS total_credits_spent_on_generat
ion
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE mp_event_name = 'generation'
GROUP BY distinct_id;
```

Bubble Purchase Spend

Event: `click_bubble_purchase`

Column: `bubble_cost`

Important Note: `bubble_cost` is a positive integer representing the credit cost of the bubble

Calculation:

`SUM(CAST(bubble_cost AS INT64))`

When calculating credit spend from bubble purchases:

- `bubble_cost` already represents a positive spend value
- No need for `ABS()`
- The sum = total credits spent on bubble purchases
- Each event records the cost of one bubble

Example query:

```
SELECT
  distinct_id,
  SUM(CAST(bubble_cost AS INT64)) AS total_credits_spent_on_bubbles
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE mp_event_name = 'click_bubble_purchase'
GROUP BY distinct_id;
```

Total Credits Spend (Combined)

To calculate total credits spent across all sources:

```
SELECT
  distinct_id,
  SUM(ABS(CAST(delta_credits AS INT64))) AS generation_spend,
  SUM(CAST(bubble_cost AS INT64)) AS bubble_spend,
  SUM(ABS(CAST(delta_credits AS INT64))) + SUM(CAST(bubble_cost AS INT
64)) AS total_spend
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE mp_event_name IN ('generation', 'click_bubble_purchase')
GROUP BY distinct_id;
```

Notes:

- Generation spend uses `delta_credits` (negative → need `ABS()`)
- Bubble spend uses `bubble_cost` (positive → no `ABS()`)
- Both represent credits deducted from user balance

Spend Logic:

- Event: 'generation'
- Column: delta_credits
- Important Note: delta_credits is a non-positive integer
- Calculation: SUM(ABS(CAST(delta_credits AS INT64)))

Remember: A negative delta_credits value (e.g., -30) means the user spent 30 credits. The ABS function converts this to a positive value for summation.

Rewards Logic

Standard Reward Events

Events that check for item_id_1/2/3 = 1 and sum corresponding quantities:

1. 'rewards_board_task'
2. 'rewards_flowers'
3. 'rewards_recipes'
4. 'rewards_scape_task'
5. 'rewards_sell_board_item'
6. 'rewards_missions_total'
7. 'rewards_missions_task'
8. 'rewards_self_collectible'
9. 'rewards_timed_task'
10. 'rewards_missions_task'
11. 'rewards_missions_total'

For these events:

- Condition: item_id_1 = 1 OR item_id_2 = 1 OR item_id_3 = 1
- Value: Sum of item_quantity_1, item_quantity_2, and item_quantity_3 where their respective item_id is 1

Special Reward Events

Harvest Collect

- Event: 'rewards_harvest_collect'
- Version-specific logic:
    - For versions < 0.35:
        - Condition: item_id_1 = 3
        - Value: item_quantity_1
    - For versions >= 0.35:
        - Condition: item_id_1 = 1
        - Value: item_quantity_1

Mass Compensation

- Event: 'rewards_mass_compensation'
- Complex logic that sums all credit rewards from multiple possible sources:
    1. mc_reward_credits (if exists)
    2. item_quantity_0 (if item_id_0 = 1)
    3. item_quantity_1 (if item_id_1 = 1)
    4. item_quantity_2 (if item_id_2 = 1)
    5. item_quantity_3 (if item_id_3 = 1)
- All sources are added together (not just first non-null)
- Null values default to 0

Example calculation:

```
sql
CopyWHEN mp_event_name = 'rewards_mass_compensation' THEN  COALESCE(CAST(mc_reward_credits AS INT64), 0) +  COALESCE(CAST(CASE WHEN item_id_0 = 1 THEN item_quantity_0 ELSE 0 END AS INT64), 0) +  COALESCE(CAST(CASE WHEN item_id_1 = 1 THEN item_quantity_1 ELSE 0 END AS INT64), 0) +  COALESCE(CAST(CASE WHEN item_id_2 = 1 THEN item_quantity_2 ELSE 0 END AS INT64), 0) +  COALESCE(CAST(CASE WHEN item_id_3 = 1 THEN item_quantity_3 ELSE 0 END AS INT64), 0)
```

Rewarded Video

- Event: 'rewards_rewarded_video'
- Simple logic:
  - Condition: item_id_1 = 1
  - Value: item_quantity_1

Version Considerations

- The 'rewards_harvest_collect' event changed in version 0.35
  - Before v0.35: Use item_id_1 = 3
  - From v0.35 onwards: Use item_id_1 = 1

General Notes

1. All quantities are cast to INT64 to ensure consistent calculations
2. COALESCE is used to handle null values, defaulting to 0
3. For all events, rewards are summed per user per day
4. Data from Ukraine (UA) and Israel (IL) is excluded from calculations

Handling FLOAT64 Fields in BigQuery

BigQuery does not allow partitioning by FLOAT64 type columns, affecting fields like `res_timestamp`, `version_float`, and `chapter`.

Quick Solutions

- **Timestamps:** `TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS event_time`
- **Versions:** `CAST(version_float AS STRING)` for partitioning (usually not needed)
- **Chapter:** Can be used directly without casting in most operations

Example Fix

```
sql
Copy-- Error: Partitioning by FLOAT64PARTITION BY distinct_id, res_timestam
p
-- Fixed: Convert to proper type firstPARTITION BY distinct_id, TIMESTAMP_M
ILLIS(CAST(res_timestamp AS INT64))
```

Always convert FLOAT64 fields to appropriate types before using them in PARTITION BY, ORDER BY, or GROUP BY clauses.

Using the `time` Field in Queries

Format and Usage

The `time` field is an alternative timestamp field available in the `vmp_master_event_normalized` table with Unix timestamp format (e.g., 1741714809.578).

Usage Guidelines:

- Use `time` only when specifically requested, otherwise default to `res_timestamp`
- Treat `time` as a TIMESTAMP data type in BigQuery - no conversion needed
- When filtering by date range: `time >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL X DAY)`
- When calculating time differences: `TIMESTAMP_DIFF(time1, time2, SECOND)`

Example:

```
SELECT  FORMAT_TIMESTAMP('%Y-%m-%d %H:%M:%S', time) AS formatte
d_time,
  TIMESTAMP_DIFF(time, LAG(time) OVER(PARTITION BY distinct_id ORDER B
Y time), SECOND) AS seconds_since_last_event
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE time >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 7 DAY)
```

Remember that `time` and `res_timestamp` may have slightly different values, so be consistent within a single analysis.

Investigating Chapter Changes

When analyzing chapter changes, especially unexpected chapter regressions, follow these best practices:

Key Investigation Guidelines

1. **Use `time` instead of `res_timestamp`** for more accurate sequence analysis
2. **Apply a minimum time threshold** (typically 1 minute) between events to exclude temporary UI state changes:

   ```
   AND TIMESTAMP_DIFF(time, prev_time, SECOND) > 60
   ```

3. **Exclude non-meaningful chapter 2 regressions**:

   ```
   AND prev_chapter != 2
   ```

4. **Filter out known false positive events** that may artificially appear as chapter regressions:

   ```
   AND mp_event_name NOT IN (
     'impression_addressables_popup_show',
     'addressable_load_start',
     'addressable_download_start',
     'addressable_download_finish',
     'algo_board_tasks_cooldown_finished')
   ```

5. **Include version information** to correlate regressions with specific app builds

Example Implementation

Use window functions with LAG() rather than self-joins for better performance:

```
WITH chapter_sequence AS (
  SELECT    distinct_id,
    chapter,
    res_timestamp,
    counter_per_session_game_side,
    version_float,
    LAG(chapter) OVER(PARTITION BY distinct_id ORDER BY res_timestamp, co
unter_per_session_game_side) AS prev_chapter,
    LAG(time) OVER(PARTITION BY distinct_id ORDER BY res_timestamp,count
er_per_session_game_side) AS prev_time
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE chapter IS NOT NULL)
SELECT *FROM chapter_sequence
WHERE  prev_chapter > chapter
  AND prev_chapter != 2  AND TIMESTAMP_DIFF(time, prev_time, SECOND) >
60  AND mp_event_name NOT IN ('impression_addressables_popup_show', /*
other excluded events */)
```

This approach provides a clean dataset for investigating true chapter regression issues while filtering out known false positives.

Avoiding Reserved Keywords in BigQuery SQL

Common Reserved Keywords to Avoid as Identifiers

When writing SQL for BigQuery, it’s important to avoid using reserved keywords as table aliases, column names, or variable names. Using reserved keywords can lead to syntax errors that may be difficult to diagnose.

Notable Reserved Keywords

- `CURRENT`: This is a reserved keyword often used in functions like `CURRENT_TIMESTAMP()`, `CURRENT_DATE()`, etc.
- `USER`: Reserved for system user information
- `TABLE`: Reserved for table definition
- `ORDER`: Used in `ORDER BY` clauses
- `GROUP`: Used in `GROUP BY` clauses
- `LIMIT`: Used to limit query results
- `TIMESTAMP`: Used for timestamp data type and functions
- `DATE`: Used for date data type and functions
- `TIME`: Used for time data type and functions
- `INTERVAL`: Used in date/time arithmetic

Best Practices

1. **Use descriptive but non-reserved names for aliases:**

```
-- IncorrectFROM events current  -- 'current' is a reserved keyword-- CorrectFROM events evt     -- 'evt' is safeFROM events curr    -- 'curr' is safe
```

2. **When using self-joins, choose clear alias names:**

```
-- RecommendedFROM events curr
JOIN events prev
```

3. **If you must use a reserved keyword, enclose it in backticks:**

```
FROM events `current`  -- Works but not recommended
```

4. **For column names that match reserved words, use aliases:**

```
SELECT  timestamp_col AS event_ts,  -- Better than 'timestamp'  date_col
AS event_date      -- Better than 'date'
```

5. **Check Error Messages:** If you get syntax errors, always look for identifiers that might be reserved keywords.

Media Source Attribution and Media Type Classification

The player’s media source represents their acquisition channel and is stored in the `first_mediasource` field of the `dim_player` table. This follows a first-touch attribution model, where each user is permanently associated with their original acquisition source.

The player's media source represents their acquisition channel and is stored in the `first_mediasource` field of the `dim_player` table. This follows a first-touch attribution model, where each user is permanently associated with their original acquisition source.

Additionally, media sources are classified into broader media types (e.g., 'social', 'offerwall', ‘networks’, 'organic', etc.) through the `sources_names_alignment` table, which provides standardized naming and categorization of acquisition channels.

```
-- Best practice for retrieving media source and media type
SELECT
  e.distinct_id,
  p.first_mediasource,
  COALESCE(sna.media_type,
    CASE
      WHEN LOWER(COALESCE(p.first_mediasource, 'organic')) = 'organic' THE
N 'organic'
      ELSE 'none'
    END
  ) AS media_type
```

```
FROM `yotam-395120.peerplay.vmp_master_event_normalized` e
LEFT JOIN `yotam-395120.peerplay.dim_player` p
  ON e.distinct_id = p.distinct_id
LEFT JOIN (
  -- Deduplicate sources_names_alignment to handle multiple entries
  SELECT
    cost_data_name,
    ARRAY_AGG(final_name ORDER BY final_name LIMIT 1)[OFFSET(0)] AS final
_name,
    ARRAY_AGG(media_type ORDER BY media_type LIMIT 1)[OFFSET(0)] AS m
edia_type
  FROM `yotam-395120.peerplay.sources_names_alignment`
  GROUP BY cost_data_name
) sna
  ON p.first_mediasource = sna.final_name
  where e.date=current_date-1
  limit 100
```

Key points:

- Always join with `dim_player` for media source information
- Use `first_mediasource` field for attribution analysis
- The media source persists throughout the user's lifetime, regardless of subsequent events

Marketing Costs Data

The data is automatically imported few times a day from Singular into `yotam-395120.singular.marketing_data` table.

1. We use the Campaign Data Schema - link
2. Updates are made in the following hours:
    a. 04:00 UTC
    b. 10:00 UTC

c. 16:00 UTC

d. 22:00 UTC

Table Schema

Category	Field Name	Description
Basic Info	date	The cost date
Basic Info	data_connector_source_name	Source name of the data connector
Basic Info	data_connector_id	Identifier for the data connector
Basic Info	data_connector_username	Username associated with the connector
Basic Info	data_connector_timestamp_utcdate	UTC timestamp of the data
Please use the date column, not this column		
App Details	app	App name as configured in Apps page
App Details	source	Name of data source (usually ad network)
App Details	os	Operating system (iOS/Android)
App Details	platform	Device platform (iPhone/iPad)
Location	country_field	User's country from network or targeting settings
Campaign Info	adn_sub_adnetwork_name	Sub-network name
Campaign Info	adn_account_id	Account ID for data source
Campaign Info	adn_account_name	Account name from ad network
Campaign Info	adn_campaign_id	Campaign ID from network

Category	Field Name	Description
Campaign Info	adn_campaign_name	Campaign name from network
Campaign Info	adn_sub_campaign_id	Sub-campaign/ad group ID
Campaign Info	adn_sub_campaign_name	Sub-campaign/ad group name
Campaign Info	adn_campaign_url	Campaign tracking link URL

Metrics

Metric Name	Description
adn_cost	Ad spend in default currency
adn_original_cost	Ad spend in original currency
adn_original_currency	Original currency from ad network
adn_impressions	Number of ad views
adn_clicks	Number of ad clicks
adn_installs	Number of app installs/conversions

2. Handling Unattributed Traffic

Unattributed traffic should be mapped to “Facebook” according to our business logic.

```
sql
Copy-- Apply business rules for media source mappingSELECT  distinct_id,
  CASE    WHEN first_media_source = 'Unattributed' THEN 'Facebook'    ELSE
first_media_source
  END AS media_source
FROM user_first_media_source
```

User Retention Calculation

Overview

Retention is a key metric for measuring user engagement over time. This document describes the methodology for calculating user retention at different intervals (D1, D3, D7, D14, D30, D60, and D90).

Definitions

- **Install Day (D0)**: The day when a user first installed the app, identified by `install_date` in `dim_player` table
- **Dx Retention**: A user is considered "retained" on day X if they had at least one event exactly X days after their install date.

Implementation

In case you need period that is already supported in `dim_player` (0, 1, 3, 7, 14, 21, 30, 45, 60, 75, 90, 120, 150, 180, 210, 240, 270, 300, 330, 360, 540, 720, 900, 1080), then use the following query:

```
-- dim_player already has pre-calculated retention flags
SELECT
  install_date,
  COUNT(DISTINCT distinct_id) AS cohort_size,
  COUNT(DISTINCT CASE WHEN d1_retention = 1 THEN distinct_id END) AS d1
_retained,
  COUNT(DISTINCT CASE WHEN d7_retention = 1 THEN distinct_id END) AS d
7_retained,
  COUNT(DISTINCT CASE WHEN d30_retention = 1 THEN distinct_id END) AS
d30_retained,

  -- Calculate retention percentages
  ROUND(100.0 * COUNT(DISTINCT CASE WHEN d1_retention = 1 THEN distin
ct_id END) / COUNT(DISTINCT distinct_id), 2) AS d1_retention_pct,
  ROUND(100.0 * COUNT(DISTINCT CASE WHEN d7_retention = 1 THEN distin
ct_id END) / COUNT(DISTINCT distinct_id), 2) AS d7_retention_pct,
  ROUND(100.0 * COUNT(DISTINCT CASE WHEN d30_retention = 1 THEN disti
nct_id END) / COUNT(DISTINCT distinct_id), 2) AS d30_retention_pct

FROM `yotam-395120.peerplay.dim_player`
WHERE
  first_country NOT IN ('UA', 'IL','AM')
  AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.f
raudsters`)
  AND install_date >= CURRENT_DATE() - 60  -- Look at last 60 days of installs
  AND install_date < CURRENT_DATE() - 30   -- Ensure D30 retention can be c
alculated
GROUP BY install_date
ORDER BY install_date DESC
```

In case the required dayX is not exist in dim_player, then use this option (in this example query it's d10):

```
WITH install_cohorts AS (
  -- Get install dates from dim_player
  SELECT
    distinct_id,
    install_date
  FROM `yotam-395120.peerplay.dim_player`
  WHERE
    first_country NOT IN ('UA', 'IL','AM')
    AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.
fraudsters`)
    AND install_date >= CURRENT_DATE() - 30  -- Look at last 30 days
    AND install_date <= CURRENT_DATE() - 10  -- Ensure D10 can be calculated
),

user_activity AS (
  -- Get activity on specific retention days
  SELECT DISTINCT
    ic.distinct_id,
    ic.install_date,
    CASE
      WHEN apd.date = DATE_ADD(ic.install_date, INTERVAL 10 DAY) THEN 1
      ELSE 0
    END AS is_d10_active
  FROM install_cohorts ic
  LEFT JOIN `yotam-395120.peerplay.agg_player_daily` apd
    ON ic.distinct_id = apd.distinct_id
    AND apd.date = DATE_ADD(ic.install_date, INTERVAL 10 DAY)
)

SELECT
  install_date,
  COUNT(DISTINCT distinct_id) AS cohort_size,
  COUNT(DISTINCT CASE WHEN is_d10_active = 1 THEN distinct_id END) AS d
10_retained,
  ROUND(100.0 * COUNT(DISTINCT CASE WHEN is_d10_active = 1 THEN distin
ct_id END) / COUNT(DISTINCT distinct_id), 2) AS d10_retention_pct
FROM user_activity
GROUP BY install_date
ORDER BY install_date DESC
```

Other cohortic calculations

dim_player supports more cohortic KPIs as retention above (for the same list of periods: 0, 1, 3, 7, 14, 21, 30, 45, 60, 75, 90, 120, 150, 180, 210, 240, 270, 300, 330, 360, 540, 720, 900, 1080)

- `dX_retention` - Retention flag (1 if active, 0 if not)
- `dX_revenue` - Total revenue by that day
- `dX_iap_revenue` - IAP revenue by that day
- `dX_ad_revenue` - Ad revenue by that day

- `dX_purchases` - Number of purchases by that day
- `dX_ads_impressions` - Number of ads watched by that day
- `dX_ftd` - First time depositor flag (1 if purchased by that day)

Connecting Normalized Event Data with Sentry Error Logs

Overview

Sentry error logs provide valuable debugging information that can be correlated with user events to understand the context and impact of errors. This section outlines the methodology for joining the normalized event data (`vmp_master_event_normalized`) with Sentry error logs (`sentry_errors`).

Sentry Table Schema

Core Identification Fields

- **`error_id`** (STRING): Individual event identifier from Sentry
- **`group_id`** (STRING): Numerical Sentry group identifier (e.g., "6868108544")
- **`issue_short_id`** (STRING): Human-readable issue ID matching Sentry UI (e.g., "MERGE-CRUISE-4FT")
- **`eventID`** (STRING): Legacy event ID field
- **`groupID`** (STRING): Legacy group ID field

Event Metadata Fields

- **`level`** (STRING): Event severity level (fatal, error, warning, info, debug)
- **`project_name`** (STRING): Sentry project name (merge-cruise, merge-cruise-ios)
- **`timestamp`** (TIMESTAMP): Event occurrence time
- **`dateCreated`** (STRING): Raw timestamp from Sentry
- **`title`** (STRING): Event title/summary

- `message` (STRING): Detailed error message

Context Fields

- `platform` (STRING): Platform (javascript, python, csharp, etc.)
- `environment` (STRING): Environment (production, staging, dev)
- `culprit` (STRING): Code location/function where error occurred
- `user_id` (STRING): User identifier for correlation with events

Join Strategy

Key Matching Fields

1. **User Identifier**: Join `distinct_id` from the event table with `user_id` from the Sentry table
2. **Timestamp Proximity**: Find the Sentry error with the closest timestamp to the event of interest
3. **Issue Tracking**: Use `issue_short_id` for human-readable issue correlation

Enhanced Join with Issue IDs

```
WITH event_data AS (
  -- Select the events of interest from normalized data
  SELECT
    distinct_id,
    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS event_time,
    res_timestamp AS event_timestamp_raw,
    -- Include any other relevant event fields
    [other_fields]
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE mp_event_name = '[YOUR_EVENT_NAME]'
    AND TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL [X] DAY)
    AND mp_country_code NOT IN ('UA', 'IL','AM')
),
sentry_match AS (
  -- Find the closest Sentry error for each event
  SELECT
    e.distinct_id,
    e.event_time,
    e.event_timestamp_raw,
    e.[other_fields],
    s.error_id,
    s.group_id,
    s.issue_short_id,  -- NEW: Human-readable issue ID
    s.level,           -- NEW: Event severity level
    s.timestamp AS sentry_timestamp,
    s.eventID,
    s.title,
    s.message,
    s.platform,
    s.environment,
    s.culprit,
    ABS(TIMESTAMP_DIFF(s.timestamp, e.event_time, SECOND)) AS time_diff,
    ROW_NUMBER() OVER(
      PARTITION BY e.distinct_id, e.event_timestamp_raw
      ORDER BY ABS(TIMESTAMP_DIFF(s.timestamp, e.event_time, SECOND))
    ) AS rn
  FROM event_data e
  LEFT JOIN `yotam-395120.peerplay.sentry_errors` s
    ON e.distinct_id = s.user_id
    -- Limit time difference to a reasonable window (adjust as needed)
    AND ABS(TIMESTAMP_DIFF(s.timestamp, e.event_time, SECOND)) <= [TIM
E_WINDOW_IN_SECONDS]
)
-- Final selection with only the closest Sentry error per event
SELECT
  sm.distinct_id,
  FORMAT_TIMESTAMP('%Y-%m-%d %H:%M:%S', sm.event_time) AS event_t
ime,
  sm.[other_fields],
  -- Sentry error information
  sm.error_id,
  sm.group_id,
  sm.issue_short_id,  -- NEW: Human-readable issue ID (MERGE-CRUISE-4FT)
  sm.level,           -- NEW: Severity level (fatal, error, warning, info, debug)
  FORMAT_TIMESTAMP('%Y-%m-%d %H:%M:%S', sm.sentry_timestamp) AS
sentry_timestamp,
  sm.eventID,
  sm.title,
  sm.message,
  sm.platform,
  sm.environment,
  sm.culprit,
  sm.time_diff AS seconds_between_event_and_error
FROM sentry_match sm
WHERE sm.rn = 1
ORDER BY sm.event_time DESC;
```

Enhanced Analytics with Issue IDs

Issue-Based Analysis

```
-- Group events by Sentry issue for impact analysis
SELECT
  s.issue_short_id,
  s.level,
  s.title,
  COUNT(DISTINCT e.distinct_id) AS affected_users,
  COUNT(*) AS total_events_near_error,
  MIN(e.event_time) AS first_event_with_error,
  MAX(e.event_time) AS last_event_with_error
FROM event_data e
JOIN `yotam-395120.peerplay.sentry_errors` s
  ON e.distinct_id = s.user_id
  AND ABS(TIMESTAMP_DIFF(s.timestamp, e.event_time, SECOND)) <= 300
```

```
WHERE s.issue_short_id IS NOT NULL
GROUP BY s.issue_short_id, s.level, s.title
ORDER BY affected_users DESC;
```

Severity-Based Event Correlation

```
-- Analyze event patterns by error severity
SELECT
  s.level AS error_severity,
  e.mp_event_name,
  COUNT(*) AS correlation_count,
  COUNT(DISTINCT s.issue_short_id) AS unique_issues,
  COUNT(DISTINCT e.distinct_id) AS affected_users
FROM event_data e
JOIN `yotam-395120.peerplay.sentry_errors` s
  ON e.distinct_id = s.user_id
  AND ABS(TIMESTAMP_DIFF(s.timestamp, e.event_time, SECOND)) <= 180
WHERE s.level IN ('fatal', 'error', 'warning')
  AND s.issue_short_id IS NOT NULL
GROUP BY s.level, e.mp_event_name
ORDER BY s.level, correlation_count DESC;
```

Issue Tracking Over Time

```
-- Track specific issues across user events
SELECT
  s.issue_short_id,
  DATE(e.event_time) AS event_date,
  COUNT(*) AS daily_correlations,
  COUNT(DISTINCT e.distinct_id) AS daily_affected_users,
  ARRAY_AGG(DISTINCT e.mp_event_name) AS related_events
FROM event_data e
JOIN `yotam-395120.peerplay.sentry_errors` s
  ON e.distinct_id = s.user_id
  AND ABS(TIMESTAMP_DIFF(s.timestamp, e.event_time, SECOND)) <= 120
```

```
WHERE s.issue_short_id = 'MERGE-CRUISE-4FT'  -- Specific issue tracking
GROUP BY s.issue_short_id, DATE(e.event_time)
ORDER BY event_date DESC;
```

Best Practices

Issue ID Utilization

- **Use `issue_short_id`** for human-readable reporting and correlation with Sentry UI
- **Use `group_id`** for programmatic joins and unique identification
- **Filter by `level`** to focus on specific severity levels (fatal, error, warning)

Handling Multiple Errors

The ROW_NUMBER() approach selects the chronologically closest error. Consider alternative approaches:

- For errors preceding events: `ORDER BY CASE WHEN s.timestamp <= e.event_time THEN s.timestamp END DESC NULLS LAST`
- For errors following events: `ORDER BY CASE WHEN s.timestamp >= e.event_time THEN s.timestamp END ASC NULLS LAST`

Additional Filtering

Consider these additional filters to improve match quality:

- **Issue-specific filters**: `WHERE s.issue_short_id LIKE 'MERGE-CRUISE-%'`
- **Severity filtering**: `WHERE s.level IN ('fatal', 'error')`
- **Environment matching**: `WHERE s.environment = 'production'`
- **Platform filtering**: `WHERE s.platform IN ('csharp', 'javascript')`

Usage Examples

Error Impact Analysis

```
-- Find all events followed by errors within 1 minute
SELECT
  event_name,
  s.level AS error_severity,
  s.issue_short_id,
  COUNT(*) AS total_events,
  COUNTIF(error_id IS NOT NULL) AS events_with_errors,
  ROUND(COUNTIF(error_id IS NOT NULL) * 100.0 / COUNT(*), 2) AS error_rat
e
FROM your_joined_data
WHERE time_diff <= 60  -- 1 minute
  AND s.issue_short_id IS NOT NULL
GROUP BY event_name, s.level, s.issue_short_id
ORDER BY error_rate DESC;
```

Critical Issue Monitoring

```
-- Monitor fatal errors and their correlation with user events
SELECT
  s.issue_short_id,
  s.title,
  COUNT(DISTINCT e.distinct_id) AS users_affected,
  COUNT(*) AS total_correlations,
  MIN(e.event_time) AS first_correlation,
  MAX(e.event_time) AS last_correlation
FROM event_data e
JOIN `yotam-395120.peerplay.sentry_errors` s
  ON e.distinct_id = s.user_id
  AND s.level = 'fatal'  -- Focus on fatal errors
  AND ABS(TIMESTAMP_DIFF(s.timestamp, e.event_time, SECOND)) <= 300
WHERE s.issue_short_id IS NOT NULL
GROUP BY s.issue_short_id, s.title
ORDER BY users_affected DESC;
```

Error Categorization by Event Context

```
-- Categorize errors by the events that preceded them
SELECT
  e.mp_event_name AS preceding_event,
  s.issue_short_id,
  s.level,
  s.platform,
  COUNT(*) AS occurrence_count,
  COUNT(DISTINCT s.group_id) AS unique_error_groups
FROM event_data e
JOIN `yotam-395120.peerplay.sentry_errors` s
  ON e.distinct_id = s.user_id
  AND s.timestamp > e.event_time  -- Error after event
  AND TIMESTAMP_DIFF(s.timestamp, e.event_time, SECOND) BETWEEN 0 A
ND 180
WHERE s.issue_short_id IS NOT NULL
GROUP BY e.mp_event_name, s.issue_short_id, s.level, s.platform
ORDER BY occurrence_count DESC;
```

Rolling Offer Credits for Value Calculations

Overview

When calculating the effective value users receive for their money ("value for money"), it's important to include credits from `rewards_rolling_offer_collect` events in addition to direct purchase rewards. This document explains why these credits should be included and how to implement this in your queries.

Rationale

Users receive certain rolling offers only after making purchases. When these offers contain credits (where `item_id = 1`), these credits should be considered part of the total value received from purchases, particularly when:

- The `sub_offer_id` is NOT 1 or 2 (these specific offers are available to non-purchasers)
- The reward includes credits (`item_id = 1`)

Including these credits provides a more accurate representation of the true value users receive for their money.

Implementation

When calculating purchase value metrics like “USD per credit,” include these additional credits in your denominator:

```
WITH rolling_offer_credits AS (
  -- Calculate credits from eligible rolling offer collect events
  SELECT
    distinct_id,
    DATE(TIMESTAMP_ADD(
      TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)),
      INTERVAL -10 HOUR
    )) AS custom_day,  -- Using 10:00 as day boundary

    -- Sum credits across all item slots
    COALESCE(CAST(CASE WHEN item_id_1 = 1 THEN item_quantity_1 ELSE 0 END AS INT64), 0) +
    COALESCE(CAST(CASE WHEN item_id_2 = 1 THEN item_quantity_2 ELSE 0 END AS INT64), 0) +
    COALESCE(CAST(CASE WHEN item_id_3 = 1 THEN item_quantity_3 ELSE 0 END AS INT64), 0) AS credit_reward

  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE
    mp_event_name = 'rewards_rolling_offer_collect'
    AND sub_offer_id NOT IN (1, 2)  -- Exclude non-purchase-contingent offers
    AND mp_country_code NOT IN ('UA', 'IL','AM')
    AND (
      (item_id_1 = 1 AND item_quantity_1 > 0) OR
      (item_id_2 = 1 AND item_quantity_2 > 0) OR
      (item_id_3 = 1 AND item_quantity_3 > 0)
    )
)
```

Value Calculation Formula

The complete USD per credit ratio should be calculated as:

```
USD per Credit = Total USD Spent / (Store Credits + Eligible Rolling Offer Cred
its)
```

Where:

- **Total USD Spent**: Sum of all purchase amounts
- **Store Credits**: Credits received directly from store purchases
- **Eligible Rolling Offer Credits**: Credits from rolling offers with sub_offer_id NOT IN (1, 2)

Usage Example

```
-- Final calculation with both credit sourcesSELECT  custom_day,
  total_usd_spent,
  total_store_credits,  -- Direct from purchases  total_rolling_offer_credits,  -- F
rom eligible rolling offers  (total_store_credits + total_rolling_offer_credits) AS t
otal_credits_received,
  CASE    WHEN (total_store_credits + total_rolling_offer_credits) > 0    THEN to
tal_usd_spent / (total_store_credits + total_rolling_offer_credits)
    ELSE NULL  END AS usd_per_credit
FROM daily_totals
WHERE (total_store_credits + total_rolling_offer_credits) > 0ORDER BY custom
_day DESC
```

Parsing of click_on_screen_raid and click_on screen

Parsing Raid Click Coordinates

Problem Context

`click_on_screen_raid` and click_on_screen have complex JSON-like string arrays that require careful parsing in BigQuery. The

Sample Structure

```
json[  {"tap_count":6},  {"target_path":"empty","timestamp":1747662318445.2
8,"x":612,"y":284},  {"target_path":"empty","timestamp":1747662318625.2
8,"x":615,"y":315},  ...  {"threshold":5}]
```

Recommended Parsing Approach

```
sql
WITH parsed_clicks AS (
  SELECT    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS timesta
mp,
    mp_event_name AS event_name,
    ARRAY(
      SELECT STRUCT(
        SAFE_CAST(JSON_VALUE(click, '$.x') AS INT64) AS x,
        SAFE_CAST(JSON_VALUE(click, '$.y') AS INT64) AS y
      )
      FROM UNNEST(SPLIT(REGEXP_REPLACE(REGEXP_REPLACE(click_on_scr
een_raid, r'^\[', ''), r'\]$', ''), '},')) AS click
      WHERE CONTAINS_SUBSTR(click, '"x":')
        AND CONTAINS_SUBSTR(click, '"y":')
    ) AS click_coordinates
```

```
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE mp_event_name = 'raid_click')
```

Converting Old Queries to Use Date Partitioning

When working with the `yotam-395120.peerplay.vmp_master_event_normalized` table, it's critical to use the date-based partitioning for efficient query execution and cost reduction. Here's how to update your old queries:

Old Pattern (Inefficient)

```
sql
-- NEFFICIENT: This pattern scans the entire table regardless of date rangeSE
LECT ...
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE DATE(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) >= CUR
RENT_DATE() - 7
```

New Pattern (Required)

```
sql
-- ✅ EFFICIENT: This pattern uses the date partition columnSELECT ...
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE date >= CURRENT_DATE() - 7
```

Key Points

1. **Always include the `date` column in your WHERE clause** when filtering by date range
2. The `date` column is our partition key and significantly improves query performance and reduces costs
3. You can still use `res_timestamp` (and TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) for time-specific filtering **in addition** to the `date` column:

```
sql
-- Correct approach with both date partition and specific time filteringSELECT
...
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE date >= CURRENT_DATE() - 7  AND TIMESTAMP_MILLIS(CAST(res_ti
mestamp AS INT64)) >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVA
L 3 HOUR)
```

Server Segments Extraction Documentation

Overview

Server segments in our game analytics data are stored in a structured format within the `server_segments` parameter. This document explains how to extract and process segment data for various game features.

Server Segments Format

The `server_segments` parameter contains data in the following format:

```
["LiveOpsData;feature1.feature2.feature3","FeatureConfigData;feature1","Item
Data;feature2"]
```

Key characteristics:

- Stored as a JSON array of strings
- Each string follows the pattern `KeyName;ValueString`
- Values may contain multiple components separated by periods (`.`)
- Common keys include: `LiveOpsData`, `ItemData`, `FeatureConfigData`

Extraction Process

To extract feature-specific segments (using Timed Tasks as an example):

1. Extract LiveOpsData Value

First, extract the full LiveOpsData string which contains feature information:

```
CASE WHEN REGEXP_CONTAINS(server_segments, r'LiveOpsData;([^"]+)')
    THEN REGEXP_EXTRACT(server_segments, r'LiveOpsData;([^"]+)')
    ELSE NULL END AS liveops_data_full
```

2. Split Into Components and Filter

Split the LiveOpsData string into individual components and filter for your target feature:

```
-- Extract all components for a feature (e.g., timed_task)(SELECT ARRAY_AGG
(component ORDER BY position)
 FROM UNNEST(SPLIT(liveops_data_full, '.')) AS component WITH OFFSET po
sition
 WHERE STARTS_WITH(component, 'timed_task')) AS feature_components
```

3. Pattern-Based Classification

For features with pattern variations (like single-digit vs. double-digit timed tasks):

```
-- Single-digit variant components (e.g., timed_task_3)(SELECT ARRAY_AGG(t
ask ORDER BY position)
 FROM UNNEST(feature_components) task WITH OFFSET position
 WHERE REGEXP_CONTAINS(task, r'timed_task_\d$')) AS single_digit_variants,
-- Double-digit variant components (e.g., timed_task_44)(SELECT ARRAY_AG
G(task ORDER BY position)
 FROM UNNEST(feature_components) task WITH OFFSET position
 WHERE REGEXP_CONTAINS(task, r'timed_task_\d\d$')) AS double_digit_varia
nts
```

4. Select Final Values with Fallbacks

Choose the appropriate components, often the last occurrence in the original sequence:

```
-- Get last single-digit variant with default fallbackCOALESCE(
  CASE    WHEN ARRAY_LENGTH(single_digit_variants) > 0    THEN single_digit
_variants[ORDINAL(ARRAY_LENGTH(single_digit_variants))]
    ELSE NULL  END,
  'default') AS primary_feature_segment
-- Get last double-digit variant with default fallbackCOALESCE(
  CASE    WHEN ARRAY_LENGTH(double_digit_variants) > 0    THEN double_di
git_variants[ORDINAL(ARRAY_LENGTH(double_digit_variants))]
    ELSE NULL  END,
  'default') AS secondary_feature_segment
```

Example: Extracting Timed Task Segments

For timed tasks, we extract both single-digit (timed_task_3) and double-digit (timed_task_44) variants from the LiveOpsData value.

Given the server_segments value:

```
["LiveOpsData;fusion_1.timed_task_5.timed_task_44.timed_task_3","FeatureCo
nfigData;fusion_1"]
```

The extraction would produce:

- primary_feature_segment: “timed_task_3” (last single-digit timed task)
- secondary_feature_segment: “timed_task_44” (last double-digit timed task)

If either pattern is missing, the value defaults to ‘default’.

This pattern can be adapted for other feature types by replacing the feature name prefix (e.g., ‘timed_task’) and adjusting the pattern matching as needed for your specific feature’s naming convention.

Mission Analytics Query Documentation

This document provides a detailed explanation of the SQL query designed to generate a daily table showing mission task completions and rewards for each user in the game. The query combines data from multiple event types and extracts valuable information from complex fields.

Overview

The query captures two main categories of mission events:

1. **Individual task completions** - When a user completes a specific mission task
2. **Total mission completions** - When a user completes an entire mission trail

For each completion event, the query pairs it with its corresponding reward event, and extracts additional metadata including constraint information and segment data.

Event Types Used

The query processes four main event types:

Event Type	Description
`missions_task_completion`	Triggered when a user completes an individual task within a mission
`rewards_missions_task`	Triggered when rewards are given for an individual task completion
`missions_total_completion`	Triggered when a user completes an entire mission trail
`rewards_missions_total`	Triggered when rewards are given for completing an entire mission trail

Key Data Extraction Techniques

1. Extracting Mission Segments

Mission segments are stored in the `server_segments` field as a JSON array of strings in the format:

```
["LiveOpsData;feature1.feature2.feature3", "MissionsConfigData;mission_2", "ItemData;feature2"]
```

The extraction process uses regex pattern matching to find and extract the MissionsConfigData value:

```
COALESCE(
  CASE WHEN REGEXP_CONTAINS(server_segments, r'MissionsConfigData;
([^"]+)')
       THEN REGEXP_EXTRACT(server_segments, r'MissionsConfigData;([^"]
+)')
       ELSE NULL END,
  'default') AS mission_segment
```

How it works:

1. `REGEXP_CONTAINS` checks if the string `MissionsConfigData;` exists in the server_segments field
2. If found, `REGEXP_EXTRACT` extracts everything after `MissionsConfigData;` until the next quotation mark
3. If not found, NULL is returned
4. `COALESCE` ensures that when no segment is found, the default value 'default' is used instead

2. Parsing Constraint Information

The `res_constraint` field contains a string that represents the constraint type and value in different formats. The query parses this string into three separate components:

1. **constraint_type**: The category of constraint
2. **constraint_value**: The primary value of the constraint
3. **min_item_level**: For chain constraints, the minimum required item level

```
-- Parse the constraint typeCASE  WHEN STARTS_WITH(res_constraint, 'item
_') THEN 'Item Constraint'  WHEN STARTS_WITH(res_constraint, 'item_level_')
THEN 'Item Level Constraint'  WHEN STARTS_WITH(res_constraint, 'chain_') T
HEN 'Chain Constraint'  WHEN STARTS_WITH(res_constraint, 'minimal_item_l
evel_') THEN 'Minimal Item Level Chain Constraint'  ELSE 'Other Constraint'EN
D AS constraint_type,
-- Parse the constraint valueCASE  WHEN STARTS_WITH(res_constraint, 'item
_')
    THEN REGEXP_EXTRACT(res_constraint, r'item_(\d+)')
  WHEN STARTS_WITH(res_constraint, 'item_level_')
    THEN REGEXP_EXTRACT(res_constraint, r'item_level_(\d+)')
  WHEN STARTS_WITH(res_constraint, 'chain_')
    THEN REGEXP_EXTRACT(res_constraint, r'chain_([^_]+)')
  WHEN STARTS_WITH(res_constraint, 'minimal_item_level_')
    THEN REGEXP_EXTRACT(res_constraint, r'minimal_item_level_\d+_chain_(\d
+)')
  ELSE NULLEND AS constraint_value,
-- Parse the minimum item levelCASE  WHEN STARTS_WITH(res_constraint, 'c
hain_')
    THEN CAST(REGEXP_EXTRACT(res_constraint, r'chain_[^_]+_(\d+)') AS INT
64)
  WHEN STARTS_WITH(res_constraint, 'minimal_item_level_')
    THEN CAST(REGEXP_EXTRACT(res_constraint, r'minimal_item_level_(\d+)_
chain') AS INT64)
  ELSE NULLEND AS min_item_level
```

Constraint formats handled:

- `item_123` - A specific item constraint (item ID 123)
- `item_level_5` - An item level constraint (minimum level 5)
- `chain_glasses_3` - A chain constraint (glasses chain, minimum level 3)
- `minimal_item_level_4_chain_501` - A minimal item level constraint for a specific chain (chain 501, minimum level 4)

3. Extracting and Combining Rewards

The query extracts reward information from both task and total reward events, with special handling for credit rewards.

```
-- Extract credit rewards (summing across all item slots where item_id = 1)CO
ALESCE(CAST(CASE WHEN item_id_1 = 1 THEN item_quantity_1 ELSE 0 END A
S INT64), 0) +COALESCE(CAST(CASE WHEN item_id_2 = 1 THEN item_quantit
y_2 ELSE 0 END AS INT64), 0) +COALESCE(CAST(CASE WHEN item_id_3 = 1
THEN item_quantity_3 ELSE 0 END AS INT64), 0) AS credit_reward,
-- Capture all rewards including non-credit items as a formatted stringARRAY_
TO_STRING(ARRAY(
  SELECT CONCAT(
    CASE WHEN item_id = 1 THEN 'Credits' ELSE CONCAT('Item_', CAST(item_i
d AS STRING)) END,
    ': ',
    CAST(quantity AS STRING)
  )
  FROM UNNEST(ARRAY[
    STRUCT(item_id_1 AS item_id, item_quantity_1 AS quantity, item_id_1_name A
S name),
    STRUCT(item_id_2 AS item_id, item_quantity_2 AS quantity, item_id_2_name
AS name),
    STRUCT(item_id_3 AS item_id, item_quantity_3 AS quantity, item_id_3_name
AS name)
  ])
  WHERE item_id IS NOT NULL AND quantity IS NOT NULL AND quantity > 0),
', ') AS reward_details
```

How reward extraction works:

1. Credit rewards (item_id = 1) are summed across all item slots to get a total credit amount
2. All rewards (including credits) are formatted into a human-readable string (e.g., “Credits: 50, Item_123: 1”)
3. Only items with non-null and non-zero quantities are included in the reward details string

Joining Completions with Rewards

The query uses two different approaches to join completion events with their corresponding reward events:

1. For Individual Task Completions

```
FROM task_completions tc
LEFT JOIN task_rewards tr
  ON tc.distinct_id = tr.distinct_id
  AND tc.missions_task_id = tr.missions_task_id
  AND tc.live_ops_id = tr.live_ops_id
  -- Join with the reward event that happened immediately after the completio
n  AND tr.res_timestamp >= tc.res_timestamp
  -- Ensure we're joining with events from the same day  AND tc.event_date = t
r.event_date
```

Joining logic for individual tasks:

1. Match the same user (`distinct_id`)
2. Match the same task (`missions_task_id`)
3. Match the same mission (`live_ops_id`)
4. Ensure the reward happened on or after the completion (time-ordered)
5. Restrict to the same calendar day to avoid incorrect matches

After joining, the query uses `MIN(reward_time)` to select only the earliest reward event that matches each completion event.

2. For Total Mission Completions

```
FROM total_completions tc
LEFT JOIN (
  SELECT    distinct_id,
    event_date,
    live_ops_id,
    MIN(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) AS reward_tim
e,
    MAX(credit_reward) AS credit_reward,
    MAX(reward_details) AS reward_details
  FROM total_rewards
  GROUP BY distinct_id, event_date, live_ops_id
) tr
ON tc.distinct_id = tr.distinct_id
AND tc.event_date = tr.event_date
AND tc.live_ops_id = tr.live_ops_id
```

Joining logic for total completions:

1. Pre-aggregate total rewards by user, date, and mission
2. Join the pre-aggregated rewards with total completions
3. Match on user, date, and mission ID rather than using timestamp ordering
4. This approach is more reliable for total completions which may have different timing patterns than individual tasks

Query Structure

The query uses a series of Common Table Expressions (CTEs) to organize the data processing:

1. `task_completions` - Extracts and processes individual mission task completion events
2. `task_rewards` - Extracts and processes rewards for individual mission tasks
3. `total_completions` - Extracts and processes total mission completion events
4. `total_rewards` - Extracts and processes rewards for total mission completions
5. `final_results` - Combines individual task completions and total completions into a single result set

The final query selects all records from `final_results` and sorts them by date, user, and timestamp, with total completions appearing after individual task completions for the same user/day.

Best Practices for Using This Query

1. **Date Range Filtering**:
   - The query includes a filter `date >= CURRENT_DATE() - 30` to limit to the last 30 days
   - Adjust this parameter based on your analysis needs to optimize performance
2. **Country Code and Currency Filtering**:
   - The query excludes data from Ukraine (UA), Israel (IL) and Aremia (AM)
   - It also excludes events with currency ‘UAH’
   - These filters follow standard practices for the game analytics
3. **Segment Handling**:
   - When a mission segment is not found, it defaults to ‘default’
   - This ensures consistent reporting even when segment data is missing
4. **Time Conversion**:
   - All timestamps are converted from Unix milliseconds (stored as FLOAT64) to proper TIMESTAMP format using `TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))`
   - This makes the output more readable and easier to use in reporting tools

Modifying the Query

Adding New Constraint Types

If new constraint formats are added to the game, update the constraint parsing CASE statements:

```
CASE  WHEN STARTS_WITH(res_constraint, 'new_format_') THEN 'New Format Constraint'  -- Add new constraint types here  ELSE 'Other Constraint'END AS constraint_type
```

Extracting Additional Segments

To extract additional segment types beyond MissionsConfigData, add similar regex patterns:

```
CASE WHEN REGEXP_CONTAINS(server_segments, r'NewSegmentType;([^"]
+)')
     THEN REGEXP_EXTRACT(server_segments, r'NewSegmentType;([^"]+)')
     ELSE NULL END AS new_segment_type
```

Time Range Adjustments

To analyze different time periods, modify the date filter:

```
-- For the last 7 daysAND date >= CURRENT_DATE() - 7-- For a specific date r
angeAND date BETWEEN '2023-01-01' AND '2023-01-31'
```

Query

```
-- Create a daily table of mission completions and rewardsWITH task_complet
ions AS (
  -- Get all mission task completions  SELECT    distinct_id,
    DATE(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) AS event_dat
e,
    res_timestamp,
    missions_task_id,
    live_ops_id,
    goal_type,
    res_constraint,
    -- Parse the constraint    CASE      WHEN STARTS_WITH(res_constraint, 'ite
m_') THEN 'Item Constraint'      WHEN STARTS_WITH(res_constraint, 'item_lev
el_') THEN 'Item Level Constraint'      WHEN STARTS_WITH(res_constraint, 'ch
ain_') THEN 'Chain Constraint'      WHEN STARTS_WITH(res_constraint, 'minim
al_item_level_') THEN 'Minimal Item Level Chain Constraint'      ELSE 'Other Co
nstraint'    END AS constraint_type,
    CASE      WHEN STARTS_WITH(res_constraint, 'item_')
        THEN REGEXP_EXTRACT(res_constraint, r'item_(\d+)')
      WHEN STARTS_WITH(res_constraint, 'item_level_')
        THEN REGEXP_EXTRACT(res_constraint, r'item_level_(\d+)')
      WHEN STARTS_WITH(res_constraint, 'chain_')
        THEN REGEXP_EXTRACT(res_constraint, r'chain_([^_]+)')
      WHEN STARTS_WITH(res_constraint, 'minimal_item_level_')
        THEN REGEXP_EXTRACT(res_constraint, r'minimal_item_level_\d+_chain_
(\d+)')
      ELSE NULL    END AS constraint_value,
    CASE      WHEN STARTS_WITH(res_constraint, 'chain_')
        THEN CAST(REGEXP_EXTRACT(res_constraint, r'chain_[^_]+_(\d+)') AS I
NT64)
      WHEN STARTS_WITH(res_constraint, 'minimal_item_level_')
        THEN CAST(REGEXP_EXTRACT(res_constraint, r'minimal_item_level_(\d
+)_chain') AS INT64)
      ELSE NULL    END AS min_item_level,
    tasks_left,
    -- Extract mission segment    COALESCE(
      CASE WHEN REGEXP_CONTAINS(server_segments, r'MissionsConfigData;
([^"]+)')
           THEN REGEXP_EXTRACT(server_segments, r'MissionsConfigData;([^"]
+)')
           ELSE NULL END,
      'default'    ) AS mission_segment
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE mp_event_name = 'missions_task_completion'    AND date >= CURR
ENT_DATE() - 30  -- Adjust time range as needed    AND mp_country_code NO
T IN ('UA', 'IL','AM')
    AND COALESCE(currency, '') != 'UAH'),
task_rewards AS (
  -- Get all mission task rewards  SELECT    distinct_id,
    DATE(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) AS event_dat
e,
    res_timestamp,
    missions_task_id,
    live_ops_id,
    -- Extract credit rewards    COALESCE(CAST(CASE WHEN item_id_1 = 1 THE
N item_quantity_1 ELSE 0 END AS INT64), 0) +    COALESCE(CAST(CASE WHE
N item_id_2 = 1 THEN item_quantity_2 ELSE 0 END AS INT64), 0) +    COALES
```

```
CE(CAST(CASE WHEN item_id_3 = 1 THEN item_quantity_3 ELSE 0 END AS IN
T64), 0) AS credit_reward,
    -- Capture other rewards (non-credit items)    ARRAY_TO_STRING(ARRAY(
      SELECT CONCAT(
        CASE WHEN item_id = 1 THEN 'Credits' ELSE CONCAT('Item_', CAST(ite
m_id AS STRING)) END,
        ': ',
        CAST(quantity AS STRING)
      )
      FROM UNNEST(ARRAY[
        STRUCT(item_id_1 AS item_id, item_quantity_1 AS quantity, item_id_1_nam
e AS name),
        STRUCT(item_id_2 AS item_id, item_quantity_2 AS quantity, item_id_2_na
me AS name),
        STRUCT(item_id_3 AS item_id, item_quantity_3 AS quantity, item_id_3_na
me AS name)
      ])
      WHERE item_id IS NOT NULL AND quantity IS NOT NULL AND quantity > 0
), ', ') AS reward_details
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE mp_event_name = 'rewards_missions_task'    AND date >= CURREN
T_DATE() - 30  -- Adjust time range as needed    AND mp_country_code NOT I
N ('UA', 'IL','AM')
    AND COALESCE(currency, '') != 'UAH'),
total_completions AS (
  -- Get all total mission completions  SELECT    distinct_id,
    DATE(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) AS event_dat
e,
    res_timestamp,
    live_ops_id,
    -- Extract mission segment    COALESCE(
      CASE WHEN REGEXP_CONTAINS(server_segments, r'MissionsConfigData;
([^"]+)')
           THEN REGEXP_EXTRACT(server_segments, r'MissionsConfigData;([^"]
+)')
           ELSE NULL END,
      'default'    ) AS mission_segment
    FROM `yotam-395120.peerplay.vmp_master_event_normalized`
    WHERE mp_event_name = 'missions_total_completion'    AND date >= CURR
ENT_DATE() - 30  -- Adjust time range as needed    AND mp_country_code NO
T IN ('UA', 'IL','AM')
      AND COALESCE(currency, '') != 'UAH'),
total_rewards AS (
    -- Get all mission total rewards  SELECT    distinct_id,
      DATE(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) AS event_dat
e,
      res_timestamp,
      live_ops_id,
      -- Extract credit rewards    COALESCE(CAST(CASE WHEN item_id_1 = 1 THE
N item_quantity_1 ELSE 0 END AS INT64), 0) +    COALESCE(CAST(CASE WHE
N item_id_2 = 1 THEN item_quantity_2 ELSE 0 END AS INT64), 0) +    COALES
CE(CAST(CASE WHEN item_id_3 = 1 THEN item_quantity_3 ELSE 0 END AS IN
T64), 0) AS credit_reward,
      -- Capture other rewards (non-credit items)    ARRAY_TO_STRING(ARRAY(
        SELECT CONCAT(
          CASE WHEN item_id = 1 THEN 'Credits' ELSE CONCAT('Item_', CAST(ite
m_id AS STRING)) END,
          ': ',
          CAST(quantity AS STRING)
        )
        FROM UNNEST(ARRAY[
          STRUCT(item_id_1 AS item_id, item_quantity_1 AS quantity, item_id_1_nam
e AS name),
          STRUCT(item_id_2 AS item_id, item_quantity_2 AS quantity, item_id_2_na
me AS name),
          STRUCT(item_id_3 AS item_id, item_quantity_3 AS quantity, item_id_3_na
me AS name)
        ])
        WHERE item_id IS NOT NULL AND quantity IS NOT NULL AND quantity > 0
), ', ') AS reward_details
    FROM `yotam-395120.peerplay.vmp_master_event_normalized`
    WHERE mp_event_name = 'rewards_missions_total'    AND date >= CURREN
```

```
T_DATE() - 30  -- Adjust time range as needed    AND mp_country_code NOT I
N ('UA', 'IL','AM')
    AND COALESCE(currency, '') != 'UAH'),
-- Combine both parts into a single unified queryfinal_results AS (
  -- Task completions with their rewards  SELECT    tc.event_date,
    tc.distinct_id,
    CAST(tc.missions_task_id AS STRING) AS missions_task_id,
    tc.live_ops_id,
    tc.goal_type,
    tc.res_constraint,
    tc.constraint_type,
    tc.constraint_value,
    tc.min_item_level,
    tc.tasks_left,
    TIMESTAMP_MILLIS(CAST(tc.res_timestamp AS INT64)) AS completion_tim
e,
    MIN(TIMESTAMP_MILLIS(CAST(tr.res_timestamp AS INT64))) AS reward_ti
me,
    MAX(tr.credit_reward) AS credit_reward,
    MAX(tr.reward_details) AS reward_details,
    FALSE AS is_total_completion,
    'Individual Task' AS completion_type,
    tc.mission_segment
  FROM task_completions tc
  LEFT JOIN task_rewards tr
    ON tc.distinct_id = tr.distinct_id
    AND tc.missions_task_id = tr.missions_task_id
    AND tc.live_ops_id = tr.live_ops_id
    -- Join with the reward event that happened immediately after the completi
on    AND tr.res_timestamp >= tc.res_timestamp
    -- Ensure we're joining with events from the same day    AND tc.event_date
= tr.event_date
  GROUP BY    tc.event_date,
    tc.distinct_id,
    tc.missions_task_id,
    tc.live_ops_id,
    tc.goal_type,
    tc.res_constraint,
    tc.constraint_type,
    tc.constraint_value,
    tc.min_item_level,
    tc.tasks_left,
    tc.mission_segment,
    tc.res_timestamp
  UNION ALL  -- Total completions with their rewards  SELECT    tc.event_date,
    tc.distinct_id,
    'TOTAL' AS missions_task_id,
    tc.live_ops_id,
    NULL AS goal_type,
    NULL AS res_constraint,
    NULL AS constraint_type,
    NULL AS constraint_value,
    NULL AS min_item_level,
    0 AS tasks_left,
    TIMESTAMP_MILLIS(CAST(tc.res_timestamp AS INT64)) AS completion_tim
e,
    tr.reward_time,
    tr.credit_reward,
    tr.reward_details,
    TRUE AS is_total_completion,
    'Total Completion' AS completion_type,
    tc.mission_segment
  FROM total_completions tc
  LEFT JOIN (
    SELECT      distinct_id,
      event_date,
      live_ops_id,
      MIN(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) AS reward_tim
e,
      MAX(credit_reward) AS credit_reward,
      MAX(reward_details) AS reward_details
    FROM total_rewards
    GROUP BY distinct_id, event_date, live_ops_id
  ) tr
  ON tc.distinct_id = tr.distinct_id
  AND tc.event_date = tr.event_date
  AND tc.live_ops_id = tr.live_ops_id
)
-- Final result sorted by date, user and timeSELECT *
FROM final_results
ORDER BY event_date DESC, distinct_id, completion_time, is_total_completion
```

Res_liveops_date Parameter

The parameter res_liveops_date in the vmp_master_event_normalized table contains the liveops_date of the event (liveops date starts at 10AM UTC, and ends in 10AM UTC in the following date). In case user asks specifically to calculate KPIs by liveops date, then use this parameter

Example Implementation - 1 (revenue in yesterday liveops date)

```
SELECT  res_liveops_date,
  SUM(CAST(price_usd AS FLOAT64)) AS total_revenue
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
WHERE  mp_event_name = 'purchase_successful'  AND CAST(price_original A
S FLOAT64) != 0.01  AND distinct_id NOT IN (SELECT distinct_id FROM `yotam
-395120.peerplay.fraudsters`)
  AND mp_country_code not in ('IL','UA','AM')
  AND currency <> 'UAH'  AND date >= current_date - 1  and res_liveops_date
= current_date - 1GROUP BY res_liveops_date
```

Example Implementation - 2 (dau in yesterday liveops date)

```
SELECT  res_liveops_date,
  count(distinct distinct_id) dau
FROM `yotam-395120.peerplay.vmp_master_event_normalized`
```

```
WHERE distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerpla
y.fraudsters`)
  and mp_country_code not in ('IL','UA','AM')
  AND date >= current_date - 1  and res_liveops_date = current_date - 1GROUP
BY res_liveops_date
```

Validate purchases in Apple Verification Service

In case analyst wants to check that Apple purchases exist in the the Apple verification service table

```
SELECT
  events.*,
  vs_request.transaction_id,
  vs_request.request_timestamp AS request_time,
  vs_approved.request_timestamp AS approved_time
FROM (
  SELECT
    date,
    distinct_id,
    purchase_funnel_id,
    request_id,
    version_float,
    price_usd,
    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS purchase_time
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE version_float >= 0.3643
    AND mp_os = 'Apple'
    AND mp_event_name = 'purchase_successful'
    AND date >= CURRENT_DATE() - 1
) events
LEFT JOIN (
  SELECT *
  FROM `yotam-395120.peerplay.verification_service_events`
  WHERE version_float >= 0.3643
    AND event_name = 'purchase_verification_request'
    AND date >= CURRENT_DATE() - 1
) vs_request
  ON vs_request.distinct_id = events.distinct_id
  AND (events.request_id=vs_request.request_id OR events.purchase_id=vs_request.transaction_id)
LEFT JOIN (
  SELECT *
  FROM `yotam-395120.peerplay.verification_service_events`
  WHERE version_float >= 0.3643
    AND event_name = 'purchase_verification_approval'
    AND date >= CURRENT_DATE() - 1
) vs_approved
  ON vs_approved.distinct_id = events.distinct_id
  AND (events.request_id=vs_approved.request_id or events.purchase_id=vs_approved.transaction_id)
WHERE vs_request.request_timestamp IS NULL
  OR vs_approved.request_timestamp IS NULL
```

KPIs only on active versions

We have table called `1`11 that contains the distinct users per each version per day and platform, and same for low payers countries.

```
WITH active_versions_filtered AS (
  -- Get versions with more than 2000 users per day  SELECT
    date,
    version_float,
    android_active_users + apple_active_users AS total_active_users
  FROM `yotam-395120.peerplay.active_versions`
  WHERE date >= CURRENT_DATE() - 7    AND (android_active_users + apple_active_users) > 1000 --filter for active versions, more than 1K users per day),
daily_metrics AS (
  SELECT    e.date,
    e.version_float,
    -- DAU (Daily Active Users)    COUNT(DISTINCT e.distinct_id) AS dau,
    -- Revenue (from purchases)    SUM(CASE
      WHEN e.mp_event_name = 'purchase_successful'
        AND CAST(e.price_original AS FLOAT64) != 0.01      THEN CAST(e.price_
usd AS FLOAT64)
      ELSE 0
    END) AS revenue
  FROM `yotam-395120.peerplay.vmp_master_event_normalized` e
  INNER JOIN active_versions_filtered avf
    ON e.date = avf.date
    AND e.version_float = avf.version_float
  WHERE
    e.date >= CURRENT_DATE() - 7    AND e.date < CURRENT_DATE()
    AND e.distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerpla
y.fraudsters`)
    AND e.mp_country_code NOT IN ('UA', 'IL','AM')
    AND COALESCE(e.currency, '') != 'UAH'  GROUP BY
    e.date,
    e.version_float
)
SELECT
  date,
  version_float,
  dau,
  ROUND(revenue, 2) AS revenue
FROM daily_metrics
ORDER BY
  date DESC,
  version_float DESC
```

Same for low payers countries active versions (they have more than 100 users in low payers countries, and less than 50 in rest of the countries)

```
WITH active_versions_filtered AS (
  -- Get versions with >500 users in low payer countries AND <500 in other co
untries  SELECT
    date,
    version_float,
    android_active_users_low_payers_countries + apple_active_users_low_paye
rs_countries AS low_payer_users,
    (android_active_users + apple_active_users) -
    (android_active_users_low_payers_countries + apple_active_users_low_pay
ers_countries) AS other_country_users
  FROM `yotam-395120.peerplay.active_versions`
  WHERE date >= CURRENT_DATE() - 14    AND (android_active_users_low_pa
yers_countries + apple_active_users_low_payers_countries) > 100    AND ((an
droid_active_users + apple_active_users) -
         (android_active_users_low_payers_countries + apple_active_users_low_p
ayers_countries)) < 50),
daily_metrics AS (
  SELECT    e.date,
    e.version_float,
    -- DAU (Daily Active Users)    COUNT(DISTINCT e.distinct_id) AS dau,
    -- Revenue (from purchases)    SUM(CASE
      WHEN e.mp_event_name = 'purchase_successful'
        AND CAST(e.price_original AS FLOAT64) != 0.01      THEN CAST(e.price_
usd AS FLOAT64)
      ELSE 0
    END) AS revenue
  FROM `yotam-395120.peerplay.vmp_master_event_normalized` e
  INNER JOIN active_versions_filtered avf
    ON e.date = avf.date
    AND e.version_float = avf.version_float
  WHERE
    e.date >= CURRENT_DATE() - 7    AND e.date < CURRENT_DATE()
    AND e.distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerpla
y.fraudsters`)
    AND e.mp_country_code NOT IN ('UA', 'IL','AM')
    AND COALESCE(e.currency, '') != 'UAH'  GROUP BY
    e.date,
    e.version_float
)
SELECT
  date,
  version_float,
  dau,
  ROUND(revenue, 2) AS revenue
FROM daily_metrics
ORDER BY
  date DESC,
  version_float DESC
```

Allocation of users per version:

```
select t1.*, (android_active_users + apple_active_users) total_users, t2.dau, rou
nd((android_active_users + apple_active_users)/t2.dau*100,2) as version_perc
entage
from yotam-395120.peerplay.active_versions t1
join
(
  select date, count(*) as dau
  from yotam-395120.peerplay.agg_player_daily
  where date = current_date - 1
  group by date
 ) t2 on t1.date=t2.date
where t1.date = current_date - 1
order by (android_active_users + apple_active_users) desc
```

Combining Game Events with Sentry Errors for User Analysis

Overview

This documentation explains how to create a unified timeline combining game events from `vmp_master_event_normalized` with error logs from `sentry_errors` for comprehensive user debugging and analysis.

Query Patterns

1. Single User Analysis

When analyzing a specific user, combine their game events and Sentry errors into a chronological timeline:

```
-- Combine game events and Sentry errors for a specific user
WITH game_events AS (
  SELECT
    distinct_id,
    CAST(chapter AS STRING) AS chapter,
    CAST(version_float AS STRING) AS version,
    mp_event_name AS event_name,
    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS event_time,
    'game_event' AS event_source,
    CAST(NULL AS STRING) AS error_id,
    CAST(NULL AS STRING) AS error_title,
    CAST(NULL AS STRING) AS error_message,
    credit_balance,
    device_id
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE distinct_id = 'YOUR_USER_ID_HERE'
    AND date >= CURRENT_DATE() - 7  -- Adjust date range as needed
),

sentry_events AS (
  SELECT
    user_id AS distinct_id,
    CAST(NULL AS STRING) AS chapter,
    CAST(NULL AS STRING) AS version,
    CONCAT('SENTRY_ERROR: ', COALESCE(title, 'Unknown Error')) AS event_name,
    timestamp AS event_time,
    'sentry_error' AS event_source,
    CAST(error_id AS STRING) AS error_id,
    title AS error_title,
    message AS error_message,
    CAST(NULL AS FLOAT64) AS credit_balance,
    CAST(NULL AS STRING) AS device_id
  FROM `yotam-395120.peerplay.sentry_errors`
  WHERE user_id = 'YOUR_USER_ID_HERE'
    AND timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 7 DAY)
)

-- Combine and order chronologically
SELECT
  distinct_id,
  chapter,
  version,
  event_name,
  event_time,
  event_source,
  error_id,
  error_title,
  error_message,
  credit_balance,
  device_id,
  FORMAT_TIMESTAMP('%Y-%m-%d %H:%M:%S', event_time) AS formatted_time
FROM (
  SELECT * FROM game_events
  UNION ALL
  SELECT * FROM sentry_events
```

```
)
ORDER BY event_time ASC;
```

2. Device-Based Analysis

When investigating issues on a specific device, find all users on that device and their combined events:

```
-- Get all users and their events/errors for a specific device
WITH device_users AS (
  SELECT DISTINCT distinct_id
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE device_id = 'YOUR_DEVICE_ID_HERE'
    AND distinct_id IS NOT NULL
),

game_events AS (
  SELECT
    e.distinct_id,
    CAST(e.chapter AS STRING) AS chapter,
    CAST(e.version_float AS STRING) AS version,
    e.mp_event_name AS event_name,
    TIMESTAMP_MILLIS(CAST(e.res_timestamp AS INT64)) AS event_time,
    'game_event' AS event_source,
    CAST(NULL AS STRING) AS error_id,
    CAST(NULL AS STRING) AS error_title,
    CAST(NULL AS STRING) AS error_message,
    e.device_id
  FROM `yotam-395120.peerplay.vmp_master_event_normalized` e
  INNER JOIN device_users du ON e.distinct_id = du.distinct_id
  WHERE e.date >= CURRENT_DATE() - 7  -- Adjust as needed
),

sentry_events AS (
  SELECT
    s.user_id AS distinct_id,
    CAST(NULL AS STRING) AS chapter,
    CAST(NULL AS STRING) AS version,
    CONCAT('SENTRY_ERROR: ', COALESCE(s.title, 'Unknown Error')) AS event
_name,
    s.timestamp AS event_time,
    'sentry_error' AS event_source,
    CAST(s.error_id AS STRING) AS error_id,
    s.title AS error_title,
    s.message AS error_message,
    CAST(NULL AS STRING) AS device_id
  FROM `yotam-395120.peerplay.sentry_errors` s
  INNER JOIN device_users du ON s.user_id = du.distinct_id
  WHERE s.timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERV
AL 7 DAY)
)

SELECT
  distinct_id,
  chapter,
  version,
  event_name,
  event_time,
  event_source,
  error_id,
  error_title,
  error_message,
  device_id,
  FORMAT_TIMESTAMP('%Y-%m-%d %H:%M:%S', event_time) AS formatted_
time
FROM (
  SELECT * FROM game_events
  UNION ALL
  SELECT * FROM sentry_events
)
ORDER BY event_time ASC, distinct_id;
```

3. Error Context Analysis

Find game events that occurred near Sentry errors to understand error context:

```
-- Find game events within a time window of Sentry errors
WITH user_errors AS (
  SELECT
    user_id AS distinct_id,
    error_id,
    timestamp AS error_time,
    title,
    message,
    eventID
  FROM `yotam-395120.peerplay.sentry_errors`
  WHERE user_id = 'YOUR_USER_ID_HERE'
    AND timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 7
DAY)
),

events_near_errors AS (
  SELECT
    e.distinct_id,
    e.mp_event_name,
    TIMESTAMP_MILLIS(CAST(e.res_timestamp AS INT64)) AS event_time,
    e.chapter,
    e.version_float,
    e.credit_balance,
    ue.error_id,
    ue.error_time,
    ue.title AS error_title,
    TIMESTAMP_DIFF(ue.error_time, TIMESTAMP_MILLIS(CAST(e.res_timestam
p AS INT64)), SECOND) AS seconds_to_error
  FROM `yotam-395120.peerplay.vmp_master_event_normalized` e
  INNER JOIN user_errors ue
    ON e.distinct_id = ue.distinct_id
    -- Events within 60 seconds before the error
    AND TIMESTAMP_MILLIS(CAST(e.res_timestamp AS INT64)) BETWEEN
      TIMESTAMP_SUB(ue.error_time, INTERVAL 60 SECOND)
      AND ue.error_time
  WHERE e.date >= CURRENT_DATE() - 7
)

SELECT
  distinct_id,
  error_id,
  error_title,
  FORMAT_TIMESTAMP('%Y-%m-%d %H:%M:%S', error_time) AS error_occur
red_at,
  mp_event_name AS preceding_event,
  FORMAT_TIMESTAMP('%Y-%m-%d %H:%M:%S', event_time) AS event_occ
urred_at,
  seconds_to_error,
  chapter,
  version_float,
  credit_balance
FROM events_near_errors
ORDER BY error_time DESC, seconds_to_error DESC;
```

Sentry Error Mapping Table

Overview

The `yotam-395120.peerplay.sentry_mapping` table provides a daily-updated mapping between Sentry error logs and game users (distinct_id), correlating errors with user activity based on IP address, device model, location, and timestamp proximity. This table enables analysis of errors in the context of user game state and progression, making it invaluable for debugging and understanding error impact on specific user segments.

Table Schema

Key columns:

- `error_id`: Unique Sentry error identifier
- `error_date`: Date when the error occurred (partition column)
- `error_time`: Formatted timestamp of the error
- `sentry_ip`, `sentry_device_model`, `sentry_city`: Error context from Sentry
- `distinct_id`: Matched game user identifier (NULL if no match found)
- `main_table_ip`, `main_device_model`, `main_city`: User context from game events
- `match_status`: 'Match Found' or 'No Match Found'
- `model_match_type`: Quality of device model match
- `total_match_score`: Confidence score of the match

Usage Examples

1. Get all errors for users in chapter 10 from yesterday

```
SELECT
  sm.error_id,
  sm.distinct_id,
  sm.error_time,
  sm.sentry_device_model,
  sm.match_status,
  sm.total_match_score
FROM `yotam-395120.peerplay.sentry_mapping` sm
INNER JOIN `yotam-395120.peerplay.agg_player_daily` apd
  ON sm.distinct_id = apd.distinct_id
  AND sm.error_date = apd.date
WHERE
  sm.error_date = CURRENT_DATE() - 1
  AND apd.last_chapter = 10
  AND sm.match_status = 'Match Found'
ORDER BY sm.error_time DESC
```

2. Find errors for high-value users (LTV > $100)

```
SELECT
  sm.error_id,
  sm.distinct_id,
  dp.ltv_revenue,
  sm.error_time,
  sm.sentry_device_model,
  sm.model_match_type
FROM `yotam-395120.peerplay.sentry_mapping` sm
INNER JOIN `yotam-395120.peerplay.dim_player` dp
  ON sm.distinct_id = dp.distinct_id
WHERE
  sm.error_date >= CURRENT_DATE() - 7
  AND dp.ltv_revenue > 100
  AND sm.match_status = 'Match Found'
ORDER BY dp.ltv_revenue DESC, sm.error_time DESC
```

3. Analyze error patterns by device model match quality

```
SELECT
  model_match_type,
  COUNT(DISTINCT error_id) as error_count,
  COUNT(DISTINCT distinct_id) as affected_users,
  AVG(total_match_score) as avg_match_score
FROM `yotam-395120.peerplay.sentry_mapping`
WHERE
  error_date >= CURRENT_DATE() - 7
  AND match_status = 'Match Found'
GROUP BY model_match_type
ORDER BY error_count DESC
```

Firebase Crashlytics Realtime Data

Overview

The `firebase_crashlytics_realtime_flattened` view provides access to crash and error data from Firebase Crashlytics for both Android and iOS platforms. This view contains rolling 30-day crash data with flattened STRUCT fields for easier querying, while preserving all array data for detailed analysis.

Table Location

```
`yotam-395120.peerplay.firebase_crashlytics_realtime_flattened`
```

Key Fields

Basic Event Information

- **event_id**: Unique identifier for the crash event
- **event_timestamp**: Timestamp when the crash/error occurred
- **received_timestamp**: Timestamp when the crash was received by Firebase
- **user_id**: User identifier (corresponds to `distinct_id` in `vmp_master_event_normalized`)
- **platform**: Operating system - 'ANDROID' or 'IOS'
- **is_fatal**: Boolean indicating if the crash was fatal

Issue Details

- **issue_id**: Unique identifier for the crash type
- **issue_title**: Title/name of the crash issue
- **issue_subtitle**: Additional details about the issue
- **error_type**: Classification of the error

Application Information

- **app_build_version**: Build version of the app

- **app_display_version**: Display version shown to users
- **bundle_identifier**: App bundle identifier

Device Information

- **device_manufacturer**: Device manufacturer (e.g., Samsung, Apple)
- **device_model**: Device model where the crash occurred
- **device_architecture**: Device processor architecture

Operating System

- **os_display_version**: OS version displayed to users
- **os_name**: Operating system name
- **os_type**: Type of operating system
- **os_device_type**: Type of device (phone, tablet, etc.)

Memory & Storage

- **memory_used**: Memory used at crash time
- **memory_free**: Memory available at crash time
- **storage_used**: Storage used
- **storage_free**: Storage available

Crash Location (Blame Frame)

- **blame_file**: File where the crash occurred
- **blame_symbol**: Symbol/function name at crash location
- **blame_line**: Line number of the crash
- **blame_library**: Library containing the crash

Unity Game Engine (if applicable)

- **unity_version**: Unity engine version
- **unity_debug_build**: Whether it's a debug build

- **unity_graphics_device_name**: Graphics card/chip name
- **unity_screen_size_px**: Screen size in pixels

Preserved Array Data

- **breadcrumbs**: Full trail of user actions before crash
- **logs**: Application logs leading to crash
- **custom_keys**: Custom key-value pairs set by the app
- **exceptions**: Array of exception details
- **errors**: Array of error details
- **threads**: Thread information at crash time

Data Retention

This table contains a **rolling 30-day window** of crash data. Older events are automatically removed.

Usage Examples

Find users with recent crashes and their game activity

```
SELECT
  fc.user_id,
  fc.platform,
  fc.device_model,
  COUNT(DISTINCT fc.issue_id) as unique_crashes,
  MAX(fc.event_timestamp) as last_crash_time,
  dp.last_chapter,
  dp.ltv_revenue
FROM `yotam-395120.peerplay.firebase_crashlytics_realtime_flattened` fc
LEFT JOIN `yotam-395120.peerplay.dim_player` dp
  ON fc.user_id = dp.distinct_id
WHERE fc.event_timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 7 DAY)
```

```
GROUP BY fc.user_id, fc.platform, fc.device_model, dp.last_chapter, dp.ltv_rev
enue
ORDER BY unique_crashes DESC;
```

Analyze crash patterns by device and OS

```
SELECT
  device_manufacturer,
  device_model,
  os_display_version,
  COUNT(DISTINCT user_id) as affected_users,
  COUNT(DISTINCT issue_id) as unique_issues,
  COUNT(*) as total_crashes,
  SUM(CASE WHEN is_fatal THEN 1 ELSE 0 END) as fatal_crashes
FROM `yotam-395120.peerplay.firebase_crashlytics_realtime_flattened`
WHERE event_timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTE
RVAL 24 HOUR)
GROUP BY device_manufacturer, device_model, os_display_version
ORDER BY total_crashes DESC;
```

Get crash details with breadcrumbs for debugging

```
SELECT
  event_timestamp,
  user_id,
  issue_title,
  blame_file,
  blame_symbol,
  blame_line,
  breadcrumbs,  -- Full breadcrumb trail preserved
  logs  -- Full logs preserved
FROM `yotam-395120.peerplay.firebase_crashlytics_realtime_flattened`
WHERE user_id = 'specific_user_id'
  AND is_fatal = TRUE
```

```
ORDER BY event_timestamp DESC
LIMIT 10;
```

Analyze Unity-specific crashes

```
SELECT
  unity_version,
  unity_graphics_device_name,
  COUNT(DISTINCT user_id) as affected_users,
  COUNT(*) as total_crashes,
  ARRAY_AGG(DISTINCT issue_title LIMIT 5) as top_issues
FROM `yotam-395120.peerplay.firebase_crashlytics_realtime_flattened`
WHERE unity_version IS NOT NULL
  AND event_timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTER
VAL 7 DAY)
GROUP BY unity_version, unity_graphics_device_name
ORDER BY total_crashes DESC;
```

Important Notes

- The `user_id` field in this table maps to `distinct_id` in the main event tables
- All STRUCT fields have been flattened into individual columns for easier querying
- All array fields (breadcrumbs, logs, exceptions, etc.) are preserved intact for detailed analysis

Unified Timeline: Combining Game Events, Crashlytics, and Sentry

Overview

When debugging complex user issues, it's often necessary to view all data sources in a single chronological timeline. This approach provides complete visibility into the user's journey, including game actions, crashes, and errors.

Unified Query Structure

The unified timeline query uses a three-step approach:

1. **Extract events from each source** into separate CTEs with standardized columns
2. **Union all events** into a single dataset
3. **Order chronologically** for timeline analysis

Standardized Output Schema

All three data sources are normalized into a common schema:

Column	Description	Source Availability
`event_time`	Timestamp of the event	All sources
`event_source`	Origin: 'game_event', 'crashlytics', or 'sentry'	All sources
`event_name`	Event/error/crash identifier	All sources
`chapter`	User's game progress	Game events only
`version`	App version	Game events, Crashlytics
`platform`	OS platform	Game events, Crashlytics
`credit_balance`	User's credit state	Game events only
`event_detail`	Context-specific information	All sources
`error_id`	Error/crash identifier	Crashlytics, Sentry
`error_title`	Error/crash description	Crashlytics, Sentry
`is_fatal`	Crash severity	Crashlytics only

Key Considerations

Time Alignment

- Game events use `TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))`
- Crashlytics uses `event_timestamp` directly

- Sentry uses `timestamp` directly
- All timestamps are converted to the same timezone for accurate ordering

Data Source Identification

Each row includes an `event_source` field to clearly identify its origin, making it easy to filter or color-code in visualization tools.

Null Handling

Fields not applicable to a data source are set to NULL with appropriate casting to maintain schema consistency across the UNION ALL operation.

Common Use Cases

1. Pre-Crash Analysis

Identify the sequence of game events leading up to a crash:

```
-- Add to WHERE clause
WHERE event_time BETWEEN
  TIMESTAMP_SUB((SELECT MIN(event_time) FROM crashlytics_events), INTE
RVAL 5 MINUTE)
  AND (SELECT MIN(event_time) FROM crashlytics_events)
```

2. Error Impact Assessment

Determine if errors correlate with specific game states:

```
-- Group by chapter and error presence
SELECT
  chapter,
  COUNTIF(event_source IN ('crashlytics', 'sentry')) as error_count,
  COUNT(*) as total_events
FROM all_events
GROUP BY chapter
```

3. Version Migration Issues

Track errors during version transitions:

```
-- Identify version changes and subsequent errors
LAG(version) OVER (PARTITION BY distinct_id ORDER BY event_time) != versi
on
```

Example: Full User Timeline Query

```
-- Unified timeline combining game events, Crashlytics crashes, and Sentry er
rors for a specific user
WITH game_events AS (
  -- Get game events from normalized table
  SELECT
    distinct_id,
    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS event_time,
    'game_event' AS event_source,
    mp_event_name AS event_name,
    CAST(chapter AS STRING) AS chapter,
    CAST(version_float AS STRING) AS version,
    mp_os AS platform,
    credit_balance,
    device_id,
    mp_model AS device_model,
    -- Game-specific fields
    CASE
      WHEN mp_event_name = 'purchase_successful' THEN CAST(price_usd A
S STRING)
      WHEN mp_event_name = 'generation' THEN CAST(delta_credits AS STRIN
G)
      WHEN mp_event_name = 'merge' THEN CONCAT(CAST(item_id_1 AS STRI
NG), ' + ', CAST(item_id_2 AS STRING))
      ELSE NULL
    END AS event_detail,
    -- Null fields for error data
    CAST(NULL AS STRING) AS error_id,
    CAST(NULL AS STRING) AS error_title,
    CAST(NULL AS STRING) AS error_message,
    CAST(NULL AS STRING) AS crash_file,
    CAST(NULL AS STRING) AS crash_line,
    CAST(NULL AS BOOLEAN) AS is_fatal,
    end_of_content,
    mode_status
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE distinct_id = '682bd823024d1aaa0945a85c'
    AND date >= CURRENT_DATE() - 30
),

crashlytics_events AS (
  -- Get crashes from Firebase Crashlytics
  SELECT
    user_id AS distinct_id,
    event_timestamp AS event_time,
    'crashlytics' AS event_source,
    CONCAT('CRASH: ', COALESCE(issue_title, 'Unknown Crash')) AS event_na
me,
    CAST(NULL AS STRING) AS chapter,
    app_display_version AS version,
    platform,
    CAST(NULL AS FLOAT64) AS credit_balance,
    CAST(NULL AS STRING) AS device_id,
    device_model,
    -- Crash-specific details
    CONCAT(
      'Type: ', COALESCE(error_type, 'N/A'),
      ' | Memory: ', CAST(memory_used AS STRING), '/', CAST(memory_free AS
STRING)
    ) AS event_detail,
    CAST(issue_id AS STRING) AS error_id,
    issue_title AS error_title,
    issue_subtitle AS error_message,
    blame_file AS crash_file,
    CAST(blame_line AS STRING) AS crash_line,
    is_fatal,
    CAST(NULL AS INT64) AS end_of_content,
    CAST(NULL AS INT64) AS mode_status
  FROM `yotam-395120.peerplay.firebase_crashlytics_realtime_flattened`
  WHERE user_id = '682bd823024d1aaa0945a85c'
    AND event_timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTE
RVAL 30 DAY)
),

sentry_events AS (
  -- Get errors from Sentry
  SELECT
    user_id AS distinct_id,
    timestamp AS event_time,
    'sentry' AS event_source,
    CONCAT('ERROR: ', COALESCE(title, 'Unknown Error')) AS event_name,
    CAST(NULL AS STRING) AS chapter,
    CAST(NULL AS STRING) AS version,
    CAST(NULL AS STRING) AS platform,
    CAST(NULL AS FLOAT64) AS credit_balance,
    CAST(NULL AS STRING) AS device_id,
    CAST(NULL AS STRING) AS device_model,
    -- Sentry-specific details
    CONCAT('EventID: ', COALESCE(eventID, 'N/A')) AS event_detail,
    CAST(error_id AS STRING) AS error_id,
    title AS error_title,
    message AS error_message,
    CAST(NULL AS STRING) AS crash_file,
    CAST(NULL AS STRING) AS crash_line,
    CAST(NULL AS BOOLEAN) AS is_fatal,
    CAST(NULL AS INT64) AS end_of_content,
    CAST(NULL AS INT64) AS mode_status
  FROM `yotam-395120.peerplay.sentry_errors`
  WHERE user_id = '682bd823024d1aaa0945a85c'
    AND timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL
30 DAY)
),

-- Combine all events
all_events AS (
  SELECT * FROM game_events
  UNION ALL
  SELECT * FROM crashlytics_events
  UNION ALL
  SELECT * FROM sentry_events
)

-- Final output with formatted timestamps and ordering
SELECT
  FORMAT_TIMESTAMP('%Y-%m-%d %H:%M:%S', event_time) AS formatted_
time,
  event_source,
  event_name,
  chapter,
  version,
  platform,
  device_model,
  credit_balance,
  end_of_content,
  mode_status,
  event_detail,
  -- Error/Crash specific fields
  error_id,
  error_title,
  error_message,
  crash_file,
  crash_line,
  is_fatal,
  -- Raw timestamp for additional processing if needed
  event_time AS raw_timestamp
FROM all_events
ORDER BY event_time DESC;
```

Users Lost State Table

Overview

The `yotam-395120.peerplay.users_lost_state` table tracks and analyzes user state loss events - situations where users experience progress regression in the game (e.g., dropping from chapter 10 back to chapter 1). This table is crucial for identifying and monitoring technical issues that cause users to lose their game progress.

Table Location

sql
`` `yotam-395120.peerplay.users_lost_state` ``

Key Fields

User and Device Identifiers

- **device_id**: Device identifier where the regression occurred
- **distinct_id_before_regression**: User ID before the state loss
- **distinct_id_after_regression**: User ID after the state loss (may differ if user identity changed)
- **latest_distinct_id**: Most recent distinct_id for this user

Chapter Progression

- **prev_chapter**: Chapter level before the regression
- **chapter_after_drop**: Chapter level immediately after the regression
- **chapter_bucket**: Grouped chapter ranges ('1', '2-3', '4-7', '8+')

- **chapter_regression_amount**: Number of chapters lost (prev_chapter - chapter_after_drop)
- **latest_chapter**: User's most recent chapter from all events
- **final_chapter**: Final chapter after any restoration attempts
- **final_gap_in_chapters**: Final progress loss after restoration attempts
- **gap_vs_latest_chapter**: Difference between prev_chapter and latest_chapter

Temporal Information

- **date**: Date of the regression event (partition column)
- **event_time**: Exact timestamp of the regression
- **prev_event_time**: Timestamp of the last event before regression
- **minutes_since_prev_event**: Time gap between events in minutes

State Restoration Analysis

- **is_distinct_id_changed**: 1 if user ID changed during regression
- **is_restore_user_state**: 1 if a restore event was triggered after regression
- **has_restore_event**: 1 if 'impression_restore_user_state_by_device_id' occurred within 150 events

Credit Balance Tracking

- **prev_credit_balance**: Credits before regression
- **credit_balance_after_drop**: Credits after regression
- **latest_credit_balance**: Most recent credit balance

User Metrics

- **install_date**: User's original install date
- **seniority**: Days since install
- **seniority_bucket**: Grouped seniority ranges ('0-7', '8-30', '30+')

Technical Details

- **mp_event_name**: Event that triggered the regression detection
- **version_float**: App version where regression occurred
- **mp_os**: Operating system (Apple/Android)

Common Queries

1. Daily State Loss Count

Get the total number of state loss events per day:

```
SELECT
  date,
  COUNT(*) AS total_state_losses,
  COUNT(DISTINCT device_id) AS affected_devices,
  COUNT(DISTINCT distinct_id_before_regression) AS affected_users,
  -- Breakdown by restoration status
  SUM(is_restore_user_state) AS restored_count,
  COUNT(*) - SUM(is_restore_user_state) AS not_restored_count,
  -- Success rate
  ROUND(100.0 * SUM(is_restore_user_state) / COUNT(*), 2) AS restoration_ra
te_pct
FROM `yotam-395120.peerplay.users_lost_state`
WHERE date >= CURRENT_DATE() - 30
GROUP BY date
ORDER BY date DESC;
```

2. Users Who Lost Progress Yesterday

Get users from yesterday whose previous chapter is higher than their latest chapter (permanent progress loss):

```
SELECT
  distinct_id_before_regression,
  device_id,
  prev_chapter,
  latest_chapter,
  prev_chapter - latest_chapter AS permanent_chapters_lost,
  chapter_after_drop,
  final_chapter,
  is_restore_user_state,
  CASE
    WHEN is_restore_user_state = 1 THEN 'Restoration Attempted'
    ELSE 'No Restoration'
  END AS restoration_status,
  version_float,
  mp_os,
  seniority_bucket
FROM `yotam-395120.peerplay.users_lost_state`
WHERE date = CURRENT_DATE() - 1
  AND prev_chapter > latest_chapter  -- Permanent progress loss
ORDER BY permanent_chapters_lost DESC, prev_chapter DESC;
```

3. Analyze State Loss by Chapter Range

Understand which chapter ranges are most affected:

```
SELECT
  chapter_bucket,
  COUNT(*) AS regression_count,
  COUNT(DISTINCT distinct_id_before_regression) AS unique_users,
  AVG(chapter_regression_amount) AS avg_chapters_lost,
  MAX(chapter_regression_amount) AS max_chapters_lost,
  SUM(is_restore_user_state) AS successful_restorations,
  ROUND(100.0 * SUM(is_restore_user_state) / COUNT(*), 2) AS restoration_success_rate
FROM `yotam-395120.peerplay.users_lost_state`
WHERE date >= CURRENT_DATE() - 7
GROUP BY chapter_bucket
ORDER BY regression_count DESC;
```

4. Track High-Value User State Losses

Monitor state losses for users with significant progress or spending:

```
SELECT
  uls.distinct_id_before_regression,
  uls.prev_chapter,
  uls.latest_chapter,
  uls.chapter_regression_amount,
  uls.is_restore_user_state,
  uls.date,
  dp.ltv_revenue,
  dp.ltv_purchases
FROM `yotam-395120.peerplay.users_lost_state` uls
LEFT JOIN `yotam-395120.peerplay.dim_player` dp
  ON uls.distinct_id_before_regression = dp.distinct_id
WHERE uls.date >= CURRENT_DATE() - 7
  AND (
    uls.prev_chapter >= 20  -- High progress users
    OR dp.ltv_revenue > 50  -- High value users
  )
ORDER BY uls.date DESC, dp.ltv_revenue DESC;
```

5. Version-Specific State Loss Analysis

State loss compare per version compared to the DAU in the last 30 days

```
WITH state_losses AS (
  SELECT
    version_float,
    mp_os,
    COUNT(*) AS state_losses,
    COUNT(DISTINCT device_id) AS affected_devices,
    AVG(chapter_regression_amount) AS avg_regression,
    SUM(CASE WHEN is_restore_user_state = 1 THEN 1 ELSE 0 END) AS restore
d,
    ROUND(100.0 * SUM(is_restore_user_state) / COUNT(*), 2) AS restore_succ
ess_pct
  FROM `yotam-395120.peerplay.users_lost_state`
  WHERE date >= CURRENT_DATE() - 30
  GROUP BY version_float, mp_os
),
active_users AS (
  SELECT
    version_float,
    mp_os,
    COUNT(DISTINCT distinct_id) AS total_active_users
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE date >= CURRENT_DATE() - 30
    AND mp_country_code NOT IN ('UA', 'IL','AM')
    AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.
fraudsters`)
  GROUP BY version_float, mp_os
)
SELECT
  sl.version_float,
  sl.mp_os,
  sl.state_losses,
  sl.affected_devices,
  au.total_active_users,
  sl.avg_regression,
  sl.restored,
  sl.restore_success_pct,
  -- Calculate state loss rate
  ROUND(100.0 * sl.state_losses / au.total_active_users, 4) AS state_loss_rate_
pct,
  ROUND(100.0 * sl.affected_devices / au.total_active_users, 4) AS affected_d
evices_rate_pct
FROM state_losses sl
LEFT JOIN active_users au
  ON sl.version_float = au.version_float
  AND sl.mp_os = au.mp_os
```

```
WHERE au.total_active_users IS NOT NULL  -- Only show versions with active users
ORDER BY sl.version_float DESC, sl.mp_os DESC;
```

Important Notes

1. **Exclusions**: The table excludes:
   - Test countries (UA, IL, AM)
   - Fraudulent users

Rate Us Prompt event

The following query extract all the users that finished board task or time board task above level 3, but didn’t get the Rate us prompt event (`impression_rating_prompt`)

```
-- Users who clicked board_tasks_go OR timed_task_go but didn't get rating prompt in last 24 hours
WITH first_click_in_target_version AS (
  -- Find each user's FIRST click (either type) in version 0.36520
  SELECT
    distinct_id,
    mp_os,
    version_float,
    MIN(TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64))) AS first_click_time,
    MIN(res_timestamp) AS first_click_timestamp_raw,
    MIN(CAST(chapter AS INT64)) AS chapter_at_click
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE
    mp_event_name IN ('click_board_tasks_go', 'click_timed_task_go')  -- Either event type
    AND version_float >= 0.3652  -- First time in THIS version
    AND mp_os = 'Apple'
    AND TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) >= TIMESTAMP
_SUB(CURRENT_TIMESTAMP(), INTERVAL 24 HOUR)
    AND CAST(chapter AS INT64) >= 4
    AND date >= CURRENT_DATE() - 1  -- Today and yesterday
    AND mp_country_code NOT IN ('UA', 'IL','AM')
    AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.
fraudsters`)
  GROUP BY distinct_id, mp_os, version_float
),

users_with_recent_rating_prompts AS (
  -- Find users who had rating prompts in the last 24 hours
  SELECT DISTINCT distinct_id
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE
    mp_event_name = 'impression_rating_prompt'
    AND date >= CURRENT_DATE() - 180  -- Today and yesterday
    AND mp_os = 'Apple'
    AND mp_country_code NOT IN ('UA', 'IL','AM')
    AND distinct_id NOT IN (SELECT distinct_id FROM `yotam-395120.peerplay.
fraudsters`)
)

-- Return users who clicked but didn't get rating prompt in last 24 hours
SELECT
  fc.distinct_id,
  fc.first_click_time,
  fc.chapter_at_click,
  fc.mp_os,
  fc.version_float
FROM first_click_in_target_version fc
LEFT JOIN users_with_recent_rating_prompts rp
  ON fc.distinct_id = rp.distinct_id
WHERE rp.distinct_id IS NULL  -- Users who didn't have rating prompts in last
24h
ORDER BY fc.first_click_time DESC
```

Google Play Sales Table

Overview

The `yotam-395120.peerplay.googleplay_sales` table contains daily imports of transaction and refunds data from the Google Play Console sales reports. This table is used for revenue reconciliation, financial reporting, and identifying discrepancies between game events and Google Play's transaction records.

Table Location

sql

`` `yotam-395120.peerplay.googleplay_sales` ``

Import Schedule

- **Frequency**: Daily collection import (every day at 2AM UTC)
- **Source**: Google Play Console sales reports
- **Data Availability**: The table doesn't contains current day data

Key Fields

Transaction Identifiers

- **order_number**: Unique Google Play order identifier (format: GPA.XXXX-XXXX-XXXX-XXXXX)
    - This field maps to `google_order_number` in the `vmp_master_event_normalized` table
- **sku_id**: Product SKU identifier (e.g., com.peerplay.mergecruise.fto002), the offer identifier (also called in-app product in Googleplay terminology)
- **product_id**: Product identifier (e.g., com.peerplay.megamerge)
- **product_type**: Type of purchase (e.g., 'inapp' for in-app purchases)

Temporal Fields

- **order_charged_date**: Date when the order was charged (DATE format)

- **order_charged_timestamp**: Unix timestamp of the charge (in seconds, not milliseconds)
- **report_month**: Month of the report (format: YYYY-MM)
- **import_date**: Timestamp when the data was imported to BigQuery

Financial Information

- **item_price**: Base price of the item in the sale currency
- **taxes_collected**: Tax amount collected
- **charged_amount**: Total amount charged (item_price + taxes_collected)
- **currency_of_sale**: Currency code (e.g., AUD, CAD, USD)
- **financial_status**: Transaction status (typically 'Charged' for completed transactions)

User Location

- **city_of_buyer**: City of the purchaser
- **state_of_buyer**: State/province of the purchaser
- **postal_code_of_buyer**: Postal code
- **country_of_buyer**: Country code (e.g., AU, CA, US)

Device Information

- **device_model**: Device model code (e.g., dm2q, r11s, a14)

Data Reconciliation

Matching with Game Events

To reconcile Google Play transactions with in-game purchase events:

```
-- Find in-game purchases that match Google Play records
SELECT e.distinct_id, e.google_order_number, e.price_usd,
g.charged_amount, g.currency_of_sale, e.mp_event_name,
TIMESTAMP_MILLIS(CAST(e.res_timestamp AS INT64)) AS game_purchase_ti
```

```
me,
TIMESTAMP_MILLIS(g.order_charged_timestamp * 1000) AS google_charge_ti
me
FROM `yotam-395120.peerplay.vmp_master_event_normalized` e
INNER JOIN `yotam-395120.peerplay.googleplay_sales` g
ON e.google_order_number = g.order_number
WHERE e.mp_event_name = 'purchase_successful' AND e.mp_os = 'Android'
AND e.date >= CURRENT_DATE() - 7
```

Finding Missing Transactions

To identify purchases recorded in-game but missing from Google Play reports:

```
WITH purchases_normalized AS (
  -- Get all Android purchases from normalized table
  SELECT
    date AS purchase_date,
    distinct_id,
    google_order_number,
    CAST(price_usd AS FLOAT64) AS price_usd,
    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS purchase_time,
    version_float,
    mp_country_code,
    currency
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE mp_event_name = 'purchase_successful'
    AND mp_os = 'Android'
    AND google_order_number IS NOT NULL
    AND google_order_number != ''
    AND CAST(price_original AS FLOAT64) != 0.01  -- Exclude test purchases
    AND mp_country_code NOT IN ('UA', 'IL','AM')
    AND COALESCE(currency, '') != 'UAH'
    AND date BETWEEN CURRENT_DATE() - 90 AND CURRENT_DATE() - 1
),
googleplay_orders AS (
  -- Get all orders from Google Play sales table
  SELECT DISTINCT
    order_number,
    order_charged_date AS order_date,
    charged_amount
  FROM `yotam-395120.peerplay.googleplay_sales`
  WHERE order_charged_timestamp IS NOT NULL
    AND order_charged_date BETWEEN CURRENT_DATE() - 90 AND CURRENT
_DATE() - 1
)
-- Find purchases that don't exist in Google Play
SELECT
  pn.purchase_date,
  pn.purchase_time,
  pn.distinct_id,
  pn.google_order_number,
  pn.price_usd,
  pn.currency,
  pn.mp_country_code,
  pn.version_float
FROM purchases_normalized pn
LEFT JOIN googleplay_orders gp
  ON pn.google_order_number = gp.order_number
WHERE gp.order_number IS NULL
ORDER BY pn.purchase_date DESC, pn.purchase_time DESC;
```

Common Use Cases

1. Daily Revenue Validation

Compare daily revenue between game events and Google Play:

```
SELECT
  order_charged_date,
  COUNT(*) AS transaction_count,
  SUM(charged_amount) AS total_charged,
  COUNT(DISTINCT SUBSTR(sku_id, 1, 30)) AS unique_products,
  COUNT(DISTINCT country_of_buyer) AS countries
FROM `yotam-395120.peerplay.googleplay_sales`
WHERE order_charged_date >= CURRENT_DATE() - 30
GROUP BY order_charged_date
ORDER BY order_charged_date DESC;
```

To identify purchases exist in Googleplay, but not in the normalized table:

```
WITH googleplay_orders AS (
  -- Get all orders from Google Play sales table
  SELECT
    order_number,
    order_charged_date AS order_date,
    TIMESTAMP_MILLIS(order_charged_timestamp * 1000) AS google_charge_ti
me,
    charged_amount,
    currency_of_sale,
    product_title,
    sku_id,
    device_model,
    country_of_buyer,
    city_of_buyer,
    financial_status
  FROM `yotam-395120.peerplay.googleplay_sales`
  WHERE order_charged_timestamp IS NOT NULL
    AND financial_status = 'Charged'  -- Only completed transactions
    AND order_charged_date BETWEEN CURRENT_DATE() - 90 AND CURRENT
_DATE() - 1
),
purchases_normalized AS (
  -- Get all Android purchases from normalized table
  SELECT DISTINCT
    google_order_number,
    date AS purchase_date,
    distinct_id,
    CAST(price_usd AS FLOAT64) AS price_usd,
    TIMESTAMP_MILLIS(CAST(res_timestamp AS INT64)) AS purchase_time
  FROM `yotam-395120.peerplay.vmp_master_event_normalized`
  WHERE mp_event_name = 'purchase_successful'
    AND mp_os = 'Android'
    AND google_order_number IS NOT NULL
    AND google_order_number != ''
    AND date BETWEEN CURRENT_DATE() - 90 AND CURRENT_DATE() - 1
)
-- Find Google Play orders that don't exist in normalized events
SELECT
  gp.order_date,
  gp.google_charge_time,
  gp.order_number,
  gp.charged_amount,
  gp.currency_of_sale,
  gp.product_title,
  gp.sku_id,
  gp.device_model,
  gp.country_of_buyer,
  gp.city_of_buyer,
  gp.financial_status
FROM googleplay_orders gp
LEFT JOIN purchases_normalized pn
  ON gp.order_number = pn.google_order_number
WHERE pn.google_order_number IS NULL  -- Order doesn't exist in normalize
d table
ORDER BY gp.order_date DESC, gp.google_charge_time DESC;
```

Important Notes

1. **Currency Considerations**: The `charged_amount` is in the `currency_of_sale`. For USD conversion, you'll need to join with exchange rate data (see example in the section below).

2. **Reconciliation Best Practices**:
    - Consider time windows when matching due to potential delays
    - Exclude test purchases (price_original = 0.01) when reconciling
3. **Missing Transactions**: Common reasons for mismatches:
    - Test or sandbox purchases are not exist in this table

Calculate Store Revenue in USD

The following query below sum up the total **usd** revenue as exist in Googleplay reports (after join with the daily currency table), and count the amount of purchases.

```
select
order_charged_date as date,
round(sum(item_price/usd_daily_rate),2) as gp_rev_after_vat_reducation,
round(sum(charged_amount/usd_daily_rate),2) as gp_rev_before_vat_reductio
n,
count(*) gp_purchaes_count
from `peerplay.googleplay_sales` gp
join `peerplay.prod_daily_currency_rates` dcr on gp.currency_of_sale=dcr.curr
ency_id and gp.order_charged_date=dcr.date
where order_charged_date>='2025-10-01'
and product_id='com.peerplay.megamerge'
and product_type='inapp'
and coalesce(usd_daily_rate,0)<>0
and financial_status='Charged'
group by order_charged_date
order by order_charged_date
```

To compare the revenue with the game data in normalized table use the following query:

```
select
*,
round((gp_rev_after_vat_reducation - game_revenue) / gp_rev_after_vat_reduc
ation * 100,2) as revenue_diff_pct,
round((gp_purchaes_count - game_purchaes_count) / gp_purchaes_count * 1
00,2) as purchases_count_diff_pct
from
(
select
order_charged_date as date, round(sum(item_price/usd_daily_rate),2) as gp_r
ev_after_vat_reducation, round(sum(charged_amount/usd_daily_rate),2) as gp_
rev_before_vat_reduction, count(*) gp_purchaes_count
--order_charged_date,item_price, taxes_collected, charged_amount, gp.curre
ncy_of_sale, dcr.*
from `peerplay.googleplay_sales` gp
join `peerplay.prod_daily_currency_rates` dcr on gp.currency_of_sale=dcr.curr
ency_id and gp.order_charged_date=dcr.date
where order_charged_date>='2025-10-01'
and product_id='com.peerplay.megamerge'
and product_type='inapp'
and coalesce(usd_daily_rate,0)<>0
and financial_status='Charged'
group by order_charged_date
order by order_charged_date
) gp
join
(
  select date, round(sum(price_usd),2) as game_revenue, count(*) as game_p
urchaes_count
  from `peerplay.vmp_master_event_normalized`
  where date>='2025-10-19'
  and mp_event_name='purchase_successful'
  and mp_os='Android'
  group by 1
) ev
on gp.date=ev.date
order by gp.date
```

Singular Events Table

Overview

The `yotam-395120.singular.singular_events` table contains exports of game events sent to Singular for attribution and analytics. This includes:

- Native Singular SDK events (e.g., `__SESSION__`, `__iap__`)
- Client-generated events via SDK (e.g., `sng_login`, `sng_level_achieved`)
- Server-to-server (S2S) events (e.g., `mc_chapter_X_complete`)

The data is updated every few hours in real-time by Singular.

Key Column Descriptions

Time Fields

- **etl_record_processing_hour_utc**: Hour when the record was received and processed by Singular (partition column), use this column as default.
- **adjusted_timestamp**: Event timestamp when it was sent from the client
- **adjusted_timestamp_unix_utc**: Unix timestamp of the adjusted timestamp

- **device_id**: Device identifier (can be IDFA, IDFV, GAID, Android ID)
- **device_id_type**: Type of device ID (idfa, idfv, gaid, android_id)
- **custom_user_id**: Game's user identifier (corresponds to `distinct_id` in game tables)
- **session_id**: Unique session identifier
- **gaid**: Google Advertising ID (Android)

- **idfa**: Identifier for Advertisers (iOS)
- **idfv**: Identifier for Vendors (iOS)
- **android_id**: Android device ID

App & Platform Information

- **platform**: Operating system (ios/android)
- **app_longname**: Full app name (e.g., com.peerplay.megamerge)
- **app_version**: Application version
- **os_version**: Operating system version
- **ip**: User's IP address

Event Details

- **name**: Event name (e.g., `__SESSION__`, `sng_level_achieved`, `mc_chapter_complete`)
- **arguments**: JSON string containing event-specific parameters (when relevant)

Revenue & Purchase Fields

- **revenue**: Revenue amount in original currency
- **converted_revenue**: Revenue converted to USD
- **currency**: Original currency code
- **converted_currency**: Converted currency (typically USD)
- **product_category**: Category of purchased product
- **is_revenue_receipt_included**: Boolean for receipt inclusion
- **is_revenue_valid**: Boolean for revenue validation

Attribution Fields

- **attribution_event_timestamp**: When user was attributed (install time)
- **attribution_event_timestamp_unix_utc**: Unix timestamp of attribution
- **touchpoint_timestamp**: Last touch point before install

- **touchpoint_timestamp_unix_utc**: Unix timestamp of touchpoint
- **is_reengagement**: Boolean for re-engagement campaigns
- **is_view_through**: Boolean for view-through attribution
- **is_fingerprinted**: Boolean for fingerprinting attribution

Location Fields

- **city**: User's city
- **country**: Country code
- **state**: State/province

Campaign & Partner Fields

- **partner**: Ad network partner (e.g., Facebook, Google, TikTok)
- **campaign_id**: Campaign identifier
- **campaign_name**: Campaign name
- **singular_campaign_name**: Singular's normalized campaign name
- **creative_id**: Creative asset identifier
- **creative_name**: Creative asset name
- **sub_campaign_id**: Sub-campaign identifier
- **sub_campaign_name**: Sub-campaign name
- **publisher_id**: Publisher identifier
- **publisher_name**: Publisher name
- **sub_publisher_id**: Sub-publisher identifier
- **sub_publisher_name**: Sub-publisher name

Common Queries

1. Event Trace for Specific User

Get all events for a specific user (distinct_id) or device identifier on a given day:

```
-- Trace events by custom_user_id (distinct_id)
SELECT adjusted_timestamp, name AS event_name, app_version, platform, ar
guments, converted_revenue, partner, campaign_name, city, country
FROM `yotam-395120.singular.singular_events`
WHERE DATE(etl_record_processing_hour_utc) = '2025-08-20' AND custom_
user_id = '68a522789403fec9e4c1faee' -- Your distinct_id
ORDER BY adjusted_timestamp ASC;
```

```


-- Trace events by IDFA or GAID
SELECT adjusted_timestamp, name AS event_name, custom_user_id, app_vers
ion, arguments, converted_revenue
FROM `yotam-395120.singular.singular_events`
WHERE DATE(etl_record_processing_hour_utc) = '2025-08-20'
AND ( idfa = '7a732efa-c6ca-4950-9ba7-086833f82971') -- iOS device OR g
aid = '3ebecb96-367a-4467-a0e1-22c9dc347792' -- Android device )
ORDER BY adjusted_timestamp ASC;
```

3. Check Event Delivery

Verify if specific events were sent to Singular in the last 14 days:

```
SELECT
  app_version,
  name,
  COUNT(*) AS num_of_events,
  COUNT(DISTINCT custom_user_id) AS unique_users
FROM `yotam-395120.singular.singular_events`
WHERE DATE(etl_record_processing_hour_utc) >= CURRENT_DATE() - 14
  AND name IN ('__iap__', 'sng_level_achieved', 'mc_chapter_complete')
GROUP BY app_version, name
ORDER BY app_version DESC, name;
```

Important Notes

1. **User ID Mapping**: The `custom_user_id` field maps to `distinct_id` in game event tables
2. **Attribution Window**: Use `attribution_event_timestamp` as the install date for cohort analysis
3. **Revenue Events**:
    - IAP revenue: `name = '__iap__'`
    - Ad revenue: `name = '__ADMON_USER_LEVEL_REVENUE__'`
4. **Organic Traffic**: Events with `partner = 'Organic'` or `partner = 'Unattributed'` represent organic installs
5. **Data Freshness**: Table updates every few hours; may not contain real-time data

Launch Funnel Analysis Table Documentation

Table Overview

The `yotam-395120.peerplay.launch_funnel_analysis_dashboard` table tracks user app launch sequences and timing, helping identify launch performance issues and failed login attempts.

Key Fields

Basic Parameters

- **distinct_id**: User identifier
- **splash_date**: Date of the launch event
- **splash_time**: Timestamp when the splash screen appeared (launch initiated)
- **full_launch_time**: Timestamp when the launch completed successfully

- **version_float**: App version
- **os**: Operating system (Apple/Android)
- **launch_type**: Type of launch event (see below)

Launch Types

The `launch_type` field has two possible values:

1. **"First Launch"**: The user's very first launch of the app after installation in this version
2. **"Repeat Launch"**: Any subsequent launch after the first

Failed Login Definition

A launch is considered **failed** when:

- `splash_to_scapes_time IS NULL`
- This means the launch sequence started (splash screen appeared) but never completed successfully
- The user never reached the main game screen

Get all failed launches from yesterday

```
SELECT
  distinct_id,
  launch_type,
  splash_time,
  version_float,
  os
FROM `yotam-395120.peerplay.launch_funnel_analysis_dashboard`
WHERE splash_to_scapes_time IS NULL
  AND splash_date = CURRENT_DATE() - 1
```

LevelPlay Impression Level Data Table

Overview

We have a daily process that import impression level data from LevelPlay to our Database.

- The data is stored in the following table (yotam-395120.peerplay.levelplay_revenue_data)
- The data is import in 12PM UTC every day and contains yesterday date + update for 2 days ago data (to complete missing data)
- Levelplay documentation link

Key Column Descriptions

Time Fields

- event_timestamp: Time (UTC) of the impression as recorded by LevelPlay
- user_id - our distinct_id
- ad_network - the network used to serve the ad
- placement - bubble or iap_store
- country - country of the user (to filter for example, UA, IL, AM)
- revenue - the estimated revenue we will get from the ad

Common use cases:

The following example compare revenue and impressions count between our game data to LevelPlay data

```
select
  lp.date,
  lp.levelplay_rv_count,
  lp.levelplay_rv_revenue,
  game.game_rv_count,
  game.game_rv_revenue,
  -- Percentage difference for RV count
  ROUND((lp.levelplay_rv_count - game.game_rv_count) / lp.levelplay_rv_count
* 100, 2) AS rv_count_diff_pct,
  -- Percentage difference for RV revenue
  ROUND((lp.levelplay_rv_revenue - game.game_rv_revenue) / lp.levelplay_rv_r
evenue * 100, 2) AS rv_revenue_diff_pct
from
(
select
date,
count(*) as levelplay_rv_count,
sum(revenue) as levelplay_rv_revenue
from yotam-395120.peerplay.levelplay_revenue_data
where date=current_date - 2
and country not in ('UA','IL','AM')
group by date
) lp
join
(
  SELECT date,
  sum(count_video_watched) AS game_rv_count,
  SUM(apd.total_ad_revenue) AS game_rv_revenue
FROM `yotam-395120.peerplay.agg_player_daily` apd
LEFT JOIN `yotam-395120.peerplay.dim_player` dp
  ON apd.distinct_id = dp.distinct_id
WHERE apd.date = CURRENT_DATE() - 2
  AND apd.total_ad_revenue > 0
  AND dp.first_country NOT IN ('UA', 'IL', 'AM')
group by date
) game
on lp.date=game.date
where (ROUND((lp.levelplay_rv_count - game.game_rv_count) / lp.levelplay_rv
_count * 100, 2)  > 2 or ROUND((lp.levelplay_rv_revenue - game.game_rv_reve
nue) / lp.levelplay_rv_revenue * 100, 2) > 2)
```