Delivery 1: KPIs

Marta Llurba, Hang Xue, Oriol Via

Code Structure

db_connect.php

```php
<?php
$servername = "localhost:3306";
$username = "hangx";
$password = "7vGPx2sehzBY";
$database = "hangx";

// Create connection
$conn = new mysqli($servername, $username, $password, $database);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>

```

Utilizes the mysqli extension to connect to the MySQL database using predefined credentials (server, username, password, database name). This is the foundation for enabling data interactions between the Unity application and the database.Includes a check for connection errors using $conn->connect_error.

Player_Data.php

```php
<?php
include 'db_connect.php';

$name = $_POST["Name"];
$country = $_POST["Country"];
$date = $_POST["Date"];

error_log("Received player data: Name={$name}, Country={$country}, Date={$date}");

$stmt = $conn->prepare("INSERT INTO `Players`(`Name`, `Country`, `Date`) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $name, $country, $date);

if ($stmt->execute()) {
    echo $conn->insert_id;
} else {
    error_log("Error in Player_Data.php: " . $stmt->error);
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

It securely inserts player data into a MySQL database using prepared statements to prevent SQL injection, and logs transactions for debugging and monitoring.

Purchase_Data.php

```php
<?php
include 'db_connect.php';

$userId = $_POST["User_ID"];
$sessionId = $_POST["Session_ID"];
$itemId = $_POST["Item"];
$buyDate = $_POST["Buy_Date"];

error_log("Received purchase data: User_ID={$userId}, Session_ID={$sessionId}, Item={$itemId}, Buy_Date={$buyDate}");

$stmt = $conn->prepare("INSERT INTO `Purchases`(`userId`, `sessionId`, `itemId`, `buyDate`) VALUES (?, ?, ?, ?)");
$stmt->bind_param("iiis", $userId, $sessionId, $itemId, $buyDate);

if ($stmt->execute()) {
    echo $conn->insert_id;
} else {
    error_log("Error in Purchase_Data.php: " . $stmt->error);
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

It efficiently handles the insertion of purchase data into a MySQL database for your Unity project, using prepared statements for security against SQL injection.

Session_Data.php

```php
<?php
include 'db_connect.php';

$userId = $_POST["User_ID"];
$startSession = $_POST["Start_Session"];

error_log("Received session start data: User_ID={$userId}, Start_Session={$startSession}");

$stmt = $conn->prepare("INSERT INTO `Sessions`(`userId`, `startSession`) VALUES (?, ?)");
$stmt->bind_param("is", $userId, $startSession);

if ($stmt->execute()) {
    echo $conn->insert_id;
} else {
    error_log("Error in Session_Data.php: " . $stmt->error);
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

It is designed for securely recording session start data into a MySQL database, utilizing prepared statements to safeguard against SQL injection.

Close_Session_Data.php

```php
<?php
include 'db_connect.php';

$sessionId = $_POST["Session_ID"];
$endSession = $_POST["End_Session"];

error_log("Received end session data: Session_ID={$sessionId}, End_Session={$endSession}");

$stmt = $conn->prepare("UPDATE `Sessions` SET `endSession` = ? WHERE `sessionId` = ?");
$stmt->bind_param("si", $endSession, $sessionId);

if ($stmt->execute()) {
    if ($stmt->affected_rows > 0) {
        //echo "Session closed successfully";
        echo $endSession;
    } else {
        error_log("No session updated in Close_Session_Data.php");
        echo "No session updated";
    }
} else {
    error_log("Error in Close_Session_Data.php: " . $stmt->error);
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

It updates session end times in a MySQL database, using prepared statements for secure data handling and providing feedback on the update status.

DataTransmission.cs

```
using System;
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;

public class DataTransmission : MonoBehaviour
{
    uint currentUserId;
    uint currentSessionId;
    uint currentPurchaseId;

    private void OnEnable()
    {
        Simulator.OnNewPlayer += HandleNewPlayer;
        Simulator.OnNewSession += HandleNewSession;
        Simulator.OnEndSession += HandleEndSession;
        Simulator.OnBuyItem += HandleBuyItem;
    }

    private void OnDisable()
    {
        Simulator.OnNewPlayer -= HandleNewPlayer;
        Simulator.OnNewSession -= HandleNewSession;
        Simulator.OnEndSession -= HandleEndSession;
        Simulator.OnBuyItem -= HandleBuyItem;
    }

    private void HandleNewPlayer(string name, string country, DateTime date)
    {
        StartCoroutine(UploadPlayer(name, country, date));
    }

    private void HandleNewSession(DateTime date)
    {
        StartCoroutine(UploadStartSession(date));
    }

    private void HandleEndSession(DateTime date)
    {
        StartCoroutine(UploadEndSession(date));
    }

    private void HandleBuyItem(int item, DateTime date)
    {
        StartCoroutine(UploadItem(item, date));
    }
```

The script subscribes to events like new player registration, session start/end, and item purchases. It unsubscribes from these events when disabled.For each event, there's a corresponding coroutine method (UploadPlayer, UploadStartSession, UploadEndSession, UploadItem) that prepares and sends the data to the server.

```
IEnumerator UploadPlayer(string name, string country, DateTime date)
{
    WWWForm form = new WWWForm();
    form.AddField("Name", name);
    form.AddField("Country", country);
    form.AddField("Date", date.ToString("yyyy-MM-dd HH:mm:ss"));

    using (UnityWebRequest www = UnityWebRequest.Post("https://citmalumnes.upc.es/~hangx/Player_Data.php", form))
    {
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            UnityEngine.Debug.LogError("Player data upload failed: " + www.error);
        }
        else
        {
            string answer = www.downloadHandler.text.Trim(new char[] { '\uFEFF', '\u200B', ' ', '\t', '\r', '\n' });
            if (uint.TryParse(answer, out uint parsedId) && parsedId > 0)
            {
                currentUserId = parsedId;
                CallbackEvents.OnAddPlayerCallback.Invoke(currentUserId);
            }
            else
            {
                UnityEngine.Debug.LogError("Invalid user ID received: " + answer);
            }
        }
    }
}
```

It uses WWWForm to create the data payload for POST requests, adding relevant fields for each type of data (player, session, purchase).It uses UnityWebRequest to send data to specific PHP endpoints (Player_Data.php, Session_Data.php, Close_Session_Data.php, Purchase_Data.php).

```
IEnumerator UploadStartSession(DateTime date)
{
    WWWForm form = new WWWForm();
    form.AddField("User_ID", currentUserId.ToString());
    form.AddField("Start_Session", date.ToString("yyyy-MM-dd HH:mm:ss"));

    string url = "https://citmalumnes.upc.es/~hangx/Session_Data.php";
    UnityWebRequest www = UnityWebRequest.Post(url, form);
    yield return www.SendWebRequest();

    if (www.result == UnityWebRequest.Result.Success)
    {
        string answer = www.downloadHandler.text.Trim(new char[] { '\uFEFF', '\u200B', ' ', '\t', '\r', '\n' });
        if (uint.TryParse(answer, out uint parsedId) && parsedId > 0)
        {
            currentSessionId = parsedId;
            CallbackEvents.OnNewSessionCallback.Invoke(currentSessionId);
        }
        else
        {
            UnityEngine.Debug.LogError("Invalid session ID received: " + answer);
        }
    }
    else
    {
        UnityEngine.Debug.LogError("Session start data upload failed: " + www.error);
    }
}


IEnumerator UploadEndSession(DateTime date)
{
    WWWForm form = new WWWForm();
    form.AddField("User_ID", currentUserId.ToString());
    form.AddField("End_Session", date.ToString("yyyy-MM-dd HH:mm:ss"));
    form.AddField("Session_ID", currentSessionId.ToString());

    string url = "https://citmalumnes.upc.es/~hangx/Close_Session_Data.php";
    UnityWebRequest www = UnityWebRequest.Post(url, form);
    yield return www.SendWebRequest();

    if (www.result == UnityWebRequest.Result.Success)
    {
        CallbackEvents.OnEndSessionCallback.Invoke(currentSessionId);
    }
    else
    {
        UnityEngine.Debug.LogError("Session end data upload failed: " + www.error);
    }

}
```

It checks for successful data transmission and logs errors if the upload fails. If successful, it parses the response (like user ID, session ID, etc.) and updates relevant fields in the script.

```
IEnumerator UploadItem(int item, DateTime date)
{
    WWWForm form = new WWWForm();
    form.AddField("Item", item.ToString());
    form.AddField("User_ID", currentUserId.ToString());
    form.AddField("Session_ID", currentSessionId.ToString());
    form.AddField("Buy_Date", date.ToString("yyyy-MM-dd HH:mm:ss"));

    UnityWebRequest www = UnityWebRequest.Post("https://citmalumnes.upc.es/~hangx/Purchase_Data.php", form);
    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        UnityEngine.Debug.LogError("Purchase data upload failed: " + www.error);
    }
    else
    {
        string answer = www.downloadHandler.text.Trim(new char[] { '\uFEFF', '\u200B', ' ', '\t', '\r', '\n'
        if (uint.TryParse(answer, out uint parsedId) && parsedId > 0)
        {
            currentPurchaseId = parsedId;
            CallbackEvents.OnItemBuyCallback.Invoke();
        }
        else
        {
            UnityEngine.Debug.LogError("Invalid purchase ID received: " + www.downloadHandler.text);
        }
    }
}
```

On successful data uploads, it invokes specific callback events to notify other parts of the Unity application.

User Activity

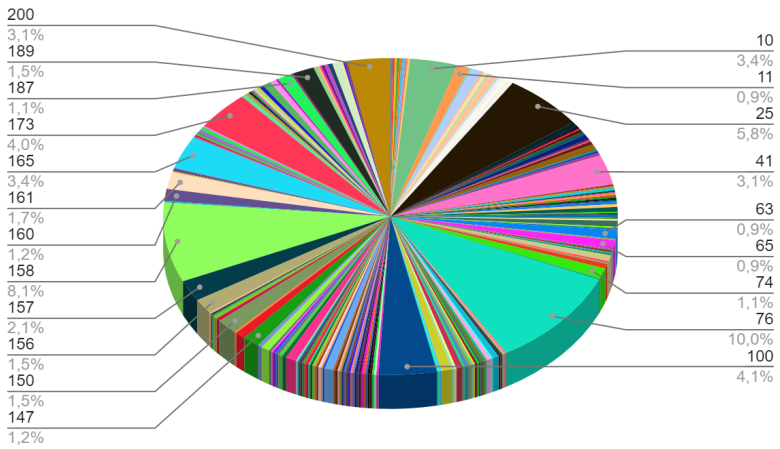
- Number of sessions per user

Data Gathered:

userId ▲ 1	num_sessions
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	22
11	6
12	1
13	5
14	1
15	2
16	1
17	4
18	1
19	1
20	1
21	2
22	1
23	1
24	1
25	38

userId	num_sessions ▼ 1
76	65
158	53
25	38
100	27
173	26
10	22
165	22
41	20
200	20
157	14
161	11
156	10
189	10
150	10
160	8
147	8
74	7
187	7
11	6
63	6
65	6
13	5
97	5
148	5
137	5

Data Explanation:



We can see that the user 76 has played the most. We have a few users playing a lot, while the majority of users. only play a few times. Those users that are playing the most might be on the top 10 of best players, since they are playing constantly.

Code:

```
SELECT
    userId,
    COUNT(sessionId) AS num_sessions
FROM Sessions
GROUP BY userId
ORDER BY num_sessions DESC
```

Code Explanation:
We retrieve the user ID from the “Sessions” table, and count the number of sessions for each user, and call that the “num_sessions”. Then we group the results by user ID, and order the result in descending order based on the “num_sessions”.

- Average session duration

Data Gathered:

avg_session_duration
56135.7603

15 hours, 35 minutes, and 35.76 seconds.

Data Explanation:
This data shows the average length of time players spend in a gaming session, helping to understand player engagement. We can see that the average is over 15 hours, which means that people are playing during large periods of time. That could show the type of game being mainly time consuming campaigns or maybe users are staying afk to farm resources.

Code:

```
SELECT
    AVG(TIMESTAMPDIFF(SECOND, startSession, endSession))
AS avg_session_duration
FROM Sessions
WHERE endSession IS NOT NULL
```

Code Explanation:
With the AVG(TIMESTAMPDIFF(SECOND, startSession, endSession)) we calculate the average session duration by finding the time difference in seconds between the start of the session and the end of the session for each session, and call that result "avg_session_duration". We also filter out the sessions where the "endSession" timestamp is NULL, so we only get the sessions that are completed.

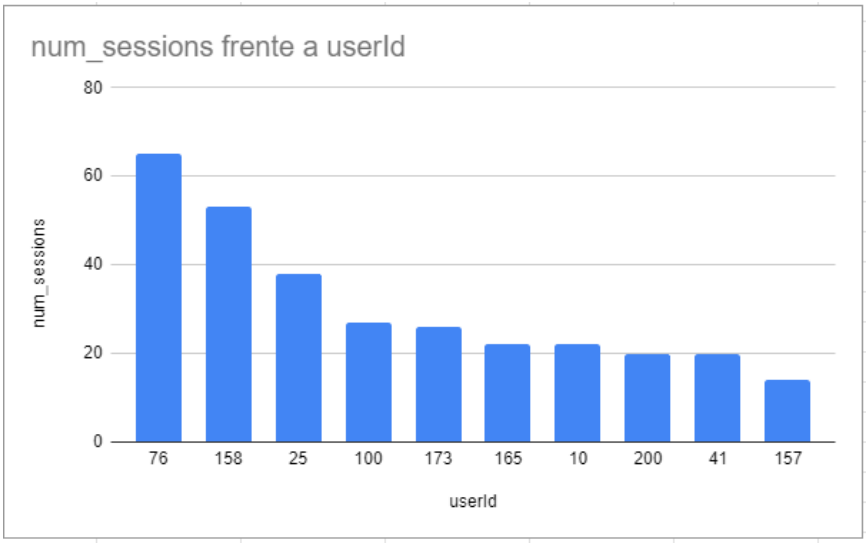
Most active users

- Based on session count

Data Gathered:

userId	num_sessions 1
76	65
158	53
25	38
100	27
173	26
165	22
10	22
200	20
41	20
157	14

Data Explanation:



When calculating the most active players based on session count, user 76 is the most active user, followed by user 158 and user 25.

The rest of users had played around an average of 20 times.

Code:

```sql
SELECT
    userId,
    COUNT(sessionId) AS num_sessions
FROM Sessions
GROUP BY userId
ORDER BY num_sessions DESC
LIMIT 10
```

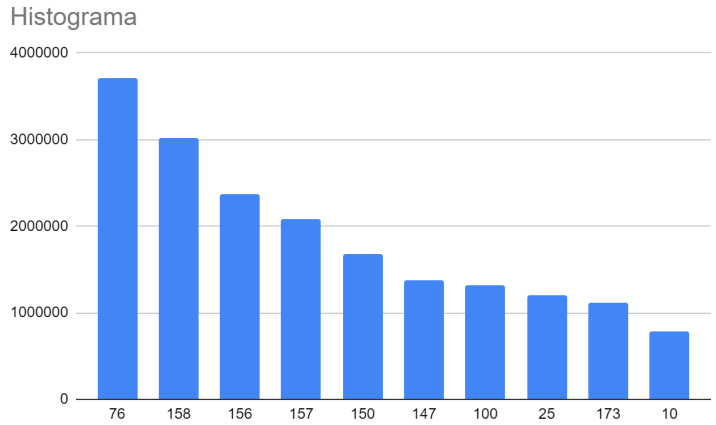
Code Explanation:
We retrieve the user ID from the “Sessions” table and count the number of sessions for each user, and call it “num_sessions”. Then, we group the results by user ID, and order them in descending order based on the “num_sessions”. We set the limit to 10, so it only shows the top 10 users.

- Based on total session duration

Data Gathered:

userId	total_duration_seconds ▾ 1
76	3704646
158	3013221
156	2377038
157	2075993
150	1682930
147	1373431
100	1313306
25	1204719
173	1110863
10	788861

Data Explanation:



When calculating the most active user based on session duration, user 76 and 158 are still the most active users, which means that both of them play in large amounts of time as well as multiple times.

Code:

```sql
SELECT
    userId,
    SUM(TIMESTAMPDIFF(SECOND, startSession, endSession))
AS total_duration_seconds
FROM Sessions
GROUP BY userId
ORDER BY total_duration_seconds DESC
LIMIT 10
```

Code Explanation:
We get the user ID from the “Sessions” table, and calculate the total duration for each user in seconds, and call it “total_duration_seconds”. We group the results by user ID, and order them descending, and limit the result to only show the first 10.

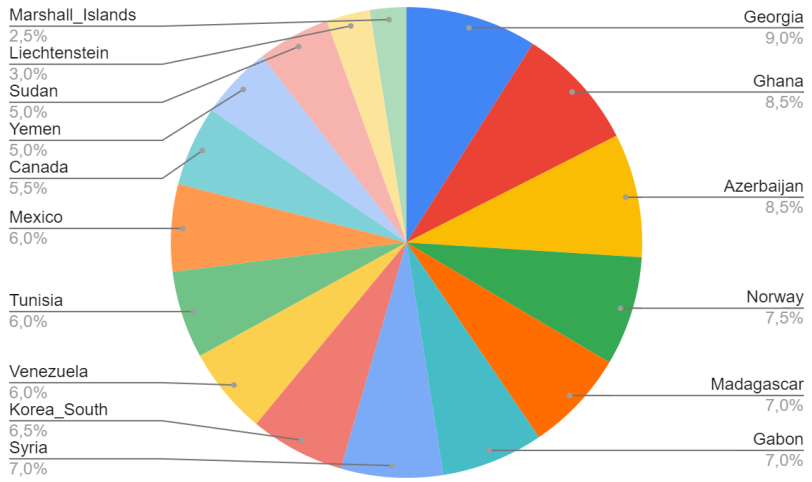
Player Demographics

- Distribution of players by country

Data Gathered:

Country	num_players ▾ 1
Georgia	18
Ghana	17
Azerbaijan	17
Norway	15
Madagascar	14
Gabon	14
Syria	14
Korea_South	13
Venezuela	12
Tunisia	12
Mexico	12
Canada	11
Yemen	10
Sudan	10
Liechtenstein	6
Marshall_Islands	5

Data Explanation:



The game is distributed mostly evenly in all the countries, Georgia being the one with the most users, having a higher reach in there.

Code:

```sql
SELECT
    Country,
    COUNT(userId) AS num_players
FROM Players
GROUP BY Country
ORDER BY num_players DESC;
```

Code Explanation:
We retrieve the country information from the “Players” table and count the number of players each country has, and call it “num_players”.

- New player acquisition rate

Data Gathered:

new_playerss	acquissition_rate
200	0.5495

Data Explanation:
We have 200 users who had only played once, up until taking this data. We have approximately 55% of new players of the total player base.

Code:

```
SELECT
    COUNT(userId) AS new_players,
    COUNT(userId) / DATEDIFF(MAX(Date), MIN(Date)) AS
acquisition_rate
FROM Players
```

Code Explanation:
COUNT(userId) AS new_players counts the total number of players, and the COUNT(userId) / DATEDIFF(MAX(Date), MIN(Date)) AS asquisition_rate calculates the new acquisition rate by dividing the total number of players by the number of days between the latest and earliest registration dates.

Purchases

- Average purchase value

Data Gathered:

num_itemss	num_purchases	average_purchase_value
5	65	13.0000

Data Explanation:
We have 5 different items, and in total, these items have been purchased 65 times, which means that, on average, each item gets purchased 13 times.

Code:

```
SELECT
    COUNT(DISTINCT p.itemId) AS num_items,
    COUNT(p.purchaseId) AS num_purchases,
    COUNT(p.purchaseId) / COUNT(DISTINCT p.itemId) AS
average_purchase_value
FROM Purchases p
```

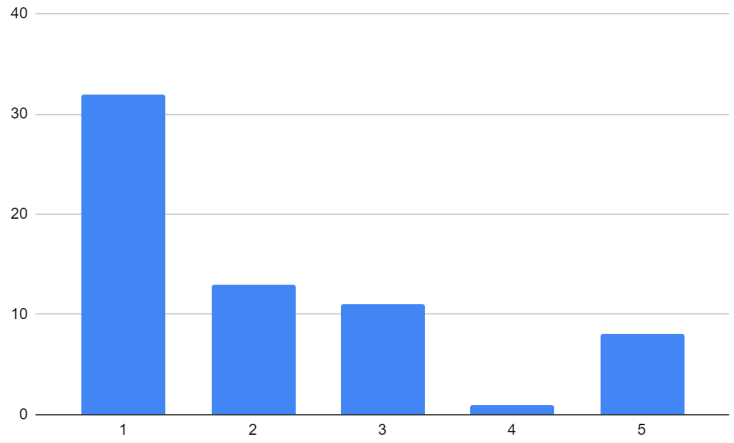
Code Explanation:
COUNT(DISTINCT p.itemId) AS num_items counts the number of unique itemsId and calls them “num_items”.
COUNT(p.purchaseId) AS num_purchases counts the total number of purchasesId and calls them “num_pruchases”.
COUNT(p.purchaseId) / COUNT(DISTINCT p.itemId) AS average_purchase_value calculates the average purchase value by dividing the total number of purchases by the number of unique items.

- Popular items

Data Gathered:

itemId	num_purchases ▼ 1
1	32
2	13
3	11
5	8
4	1

Data Explanation:
With this data we can see that item 1 is the most popular by a lot, while item 4 has been purchased only 1 time. We can deduce that item 1 might be the best item, most useful or it is part of a temporary event, and users are purchasing it before this event is over.



Code:

```sql
SELECT
    itemId,
    COUNT(purchaseId) AS num_purchases
FROM Purchases
GROUP BY itemId
ORDER BY num_purchases DESC
```

Code Explanation:
COUNT(purchaseId) AS num_purchases counts the number of purchases for each item and calls it “num_purchases”, and we order them in descending order.

User Engagement

- Conversion rate from sessions to purchases

Data Gathered:

num_sessionss	num_purchases	conversion_rate
653	65	0.0995

Data Explanation:
In a total of 653 sessions, there have been 65 purchases, which means that almost 1% of the sessions get a purchase from the users.

Code:

```
SELECT
    COUNT(DISTINCT s.sessionId) AS num_sessions,
    COUNT(DISTINCT p.purchaseId) AS num_purchases,
    COUNT(DISTINCT p.purchaseId) / COUNT(DISTINCT
s.sessionId) AS conversion_rate
FROM Sessions s
LEFT JOIN Purchases p ON s.sessionId = p.sessionId
```

Code Explanation:
COUNT(DISTINCT s.sessionId) AS num_sessions counts the number of distinct sessions
COUNT(DISTINCT p.purchaseId) AS num_purchases counts the number of distinct purchases
COUNT(DISTINCT p.purchaseId) / COUNT(DISTINCT s.sessionId) AS conversion_rate calculates the conversion rate by dividing the number of purchases by the number of sessions

- Retention rate (how many users return for multiple sessions)

Data Gathered:

num_returning_users	num_users_on_first_session	retention_rate
31	31	1.0000

Data Explanation:
All 31 users who had their first session returned for additional sessions. The retention rate of 1 indicates a perfect retention scenario, where every user came back for more sessions.

Code:

```sql
SELECT
    COUNT(DISTINCT r1.userId) AS num_returning_users,
    COUNT(DISTINCT r2.userId) AS
num_users_on_first_session,
    COUNT(DISTINCT r1.userId) / COUNT(DISTINCT r2.userId)
AS retention_rate
FROM Sessions r1
JOIN Sessions r2 ON r1.userId = r2.userId
    AND r1.sessionId <> r2.sessionId
    AND r1.startSession < r2.startSession
```

Code Explanation:
COUNT(DISTINCT r1.userId) AS num_returning_users counts the number of distinct users who have returned for multiple sessions.
COUNT(DISTINCT r2.userId) AS num_users_on_first_session counts the number of distinct users on their first session.
COUNT(DISTINCT r1.userId) / COUNT(DISTINCT r2.userId) AS retention_rate calculates the retention rate by dividing the number of returning users by the number of users on their first session.
FROM Sessions r1 specifies the first instance of the “Sessions” table and calls it “r1”.
JOIN Sessions r2 ON r1.userId = r2.userId performs a self-join with the “Sessions” table for the second instance and calls it “r2”.
AND r1.sessionId <> r2.sessionId & AND r1.startSession < r2.startSession specifies the conditions for joining, ensuring that the sessions are different, and the second session occurs after the first.

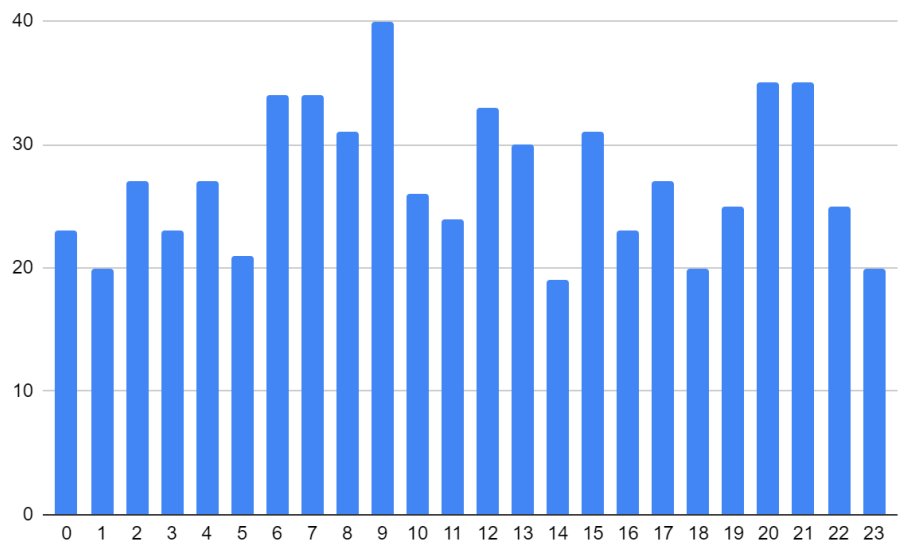
Session Analysis

- Peak usage hours

Data Gathered:

hour_of_day ▲ 1	num_sessions
0	23
1	20
2	27
3	23
4	27
5	21
6	34
7	34
8	31
9	40
10	26
11	24
12	33
13	30
14	19
15	31
16	23
17	27
18	20
19	25
20	35
21	35
22	25
23	20

Data Explanation:
We can see that the most played hours in the day are at 9 am, and at 20 - 21 pm. This shows that users play after waking up and before going to sleep.



Code:

```
SELECT
    HOUR(startSession) AS hour_of_day,
    COUNT(sessionId) AS num_sessions
FROM Sessions
GROUP BY hour_of_day
ORDER BY `hour_of_day` ASC LIMIT 24
```

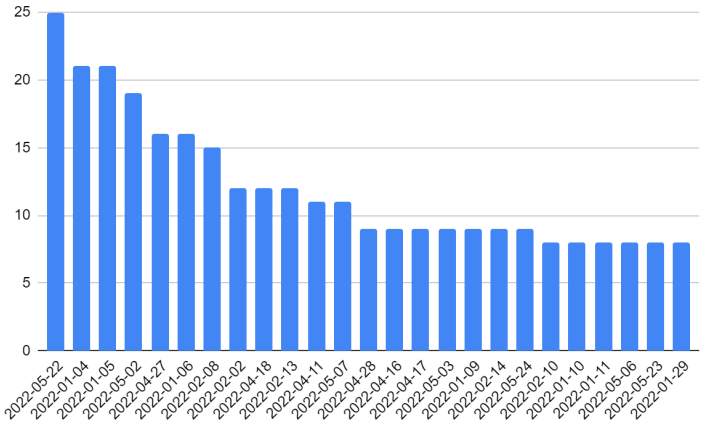
Code Explanation:
HOUR(startSession) AS hour_of_day extracts the hour from the “sessionStart” timestamp and we call it “hour_of_day”.
COUNT(sessionId) AS num_sessions counts the number of sessions for each hour.

- Peak usage days

Data Gathered:

day	num_sessions ▾ 1
2022-05-22	25
2022-01-04	21
2022-01-05	21
2022-05-02	19
2022-04-27	16
2022-01-06	16
2022-02-08	15
2022-02-02	12
2022-04-18	12
2022-02-13	12
2022-04-11	11
2022-05-07	11
2022-04-28	9
2022-04-16	9
2022-04-17	9
2022-05-03	9
2022-01-09	9
2022-02-14	9
2022-05-24	9
2022-02-10	8
2022-01-10	8
2022-01-11	8
2022-05-06	8
2022-05-23	8
2022-01-29	8

Data Explanation:



The most played day is the 22nd of May, but we can see that there is a high player count in early January (04 , 05, 06) which indicates that during those days, users have played more. This could be due to christmas, and being the last few days before going back to work or school, and also meaning that the game might have a special holiday event during those 3 days.

Code:

```
SELECT
    DATE(startSession) AS day,
    COUNT(sessionId) AS num_sessions
FROM Sessions
GROUP BY day
ORDER BY `num_sessions`  DESC LIMIT 25
```

Code Explanation:
DATE(startSession) AS day extracts the date from the “startSession” timestamp and calls it “day”.
COUNT(sessionId) AS num_sessions counts the number of sessions for each day.

- Average time between sessions

Data Gathered:

avg_time_between_sessions
248833.2176

248833.2176 seconds = 69.12 hours

Data Explanation:
The average time between the end of one session, and the start of the next one is 69.12 hours, which means that, on average, people play every 3 days more or less. This could be that, dues having large periods of time playing, they need large periods of time of break.

Code:

```
SELECT
    AVG(TIMESTAMPDIFF(SECOND, s1.endSession,
s2.startSession)) AS avg_time_between_sessions
FROM Sessions s1
JOIN Sessions s2 ON s1.userId = s2.userId
    AND s1.sessionId <> s2.sessionId
    AND s1.endSession < s2.startSession
```

Code Explanation:
VG(TIMESTAMPDIFF(SECOND, s1.endSession, s2.startSession)) AS avg_time_between_sessions calculates the average time between sessions by finding the time difference in seconds between the end of the first session “s1.endSession”, and the start of the second session “s2.startSession”.
AND s1.sessionId <> s2.sessionId & AND s1.endSession < s2.startSession specifies the conditions for joining, ensuring that the sessions are different and the second session starts after the first ends.