Report

Before discussing all the solutions, I’d like to claim that all the queries are designed with consistent session\_id, all samples(rows) with same session\_id where “gameEnded” time is earlier that “gameStarted” time are not considered. Such as example below:

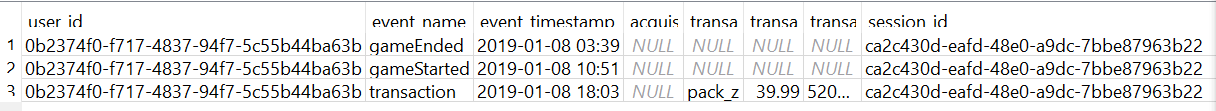
**Query:**

select \*

from events

where user\_id like "0b2374f%" AND session\_id like "ca2c%";

**Output:**



In this very example, all the queries will ignore the above records. Notice this will affect the results such as daily active user, weekly active user, average time played, etc.

Solutions

1.a: see queries in .sql file

1.b: see queries in .sql file

1.c:

Here, I used Falcon SQL client for query visualization, **Fig.1** is the bar plot for daily active use, the x axis represents date, and each bar stands for the unique number of players that played on the day. **Fig.2** shows the bars where orange ones are active user in the corresponding week, while blue bars represents daily active user**. Fig.3** shows the weekly stickiness per day (DAU/WAU).

**A screenshot of a cell phone

Description automatically generated**

**Fig1. Bar plot for daily active user**

A picture containing writing implement, stationary, pencil

Description automatically generated

**Fig2.Bar plot for weekly active user and daily active user.**

A close up of text on a white background

Description automatically generated

**Fig.3 Line plot for weekly stickiness**

2.a: see queries in .sql file

2.b: see queries in .sql file

2.c:

Here, I used Falcon SQL client for query visualization, **Fig.4** is the bar plot for daily revenue, the x axis represents date, and each bar stands for the overall revenue on the day. **Fig.5** shows the daily conversion rate per day calculated by unique users that made purchases / daily active user.

A screenshot of a cell phone

Description automatically generated

**Fig.4 Bar plot for daily revenue**

A screenshot of a cell phone

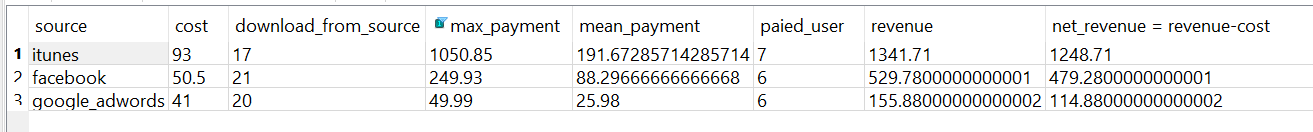
Description automatically generated

**Fig.5 Bar plot for daily conversion rate**

3: see queries in .sql file

4.a: see queries in .sql file

4.b:

First, i wrote a query that returns the revenue table as **Table.1**. 

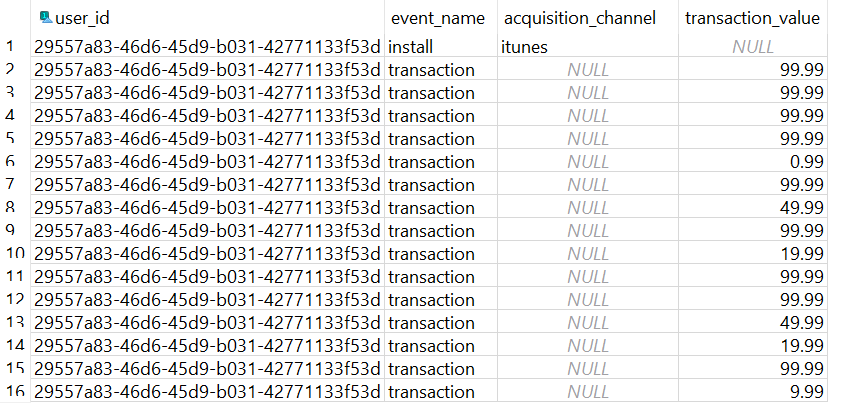
**Table1. The revenue of each channel.** The table collected the overall cost for advertising, amount of people that download through the channel, the overall max payment one user made, number of users that made a purchase, overall revenue and net revenue (revenue - coat) during 2019-01-01 to 2019-01-31. E.g. 17 people downloaded the game through iTunes, costs us 93 dollars for the installations, 7 out of these 17 people made purchases, the max revenue from one of these 7 payed user is 1050.85 dollars, while the average revenue of all 7 payed users is 191.7, the overall revenue made by these 7 payed user is 1341.71, the net revenue (revenue - cost) is 1248 dollars. The **Fig.6** shows the net revenue, mean and max payment users made in January per channel.

A screenshot of a cell phone

Description automatically generated

**Fig.6 Revenue bars per channel.** Blues ones are net revenue, calculated by revenue – cost, oranges bars are mean payment made by payed user, max payment by one of all payed users is shown as green bars. For example, in January on channel google AdWords (first group), we earned 114.88 dollars (revenue - cost) by advertising, the mean payment by google users in this month is 25.98, the max payment one google user made is 50 dollars in this month.

We can observe that though advertising on iTunes costs us most (93 dollars), and the purchase users proportion(payed user / download-from-channel user) has no big difference than other channel (7/17 versus 6/21 and 6/20), it does bring us users such as “user\_id: 29557a83-46d6-45d9-b031-42771133f53d” (**Table.2**)that make big purchases, the overall payment by this user in January is 1050.85 dollars. Thus, I suggest we focus on iTunes for advising.



**Table 2. Purchases table by iTunes user “29557a83-46d6-45d9-b031-42771133f53d”**

5. see queries in .sql file

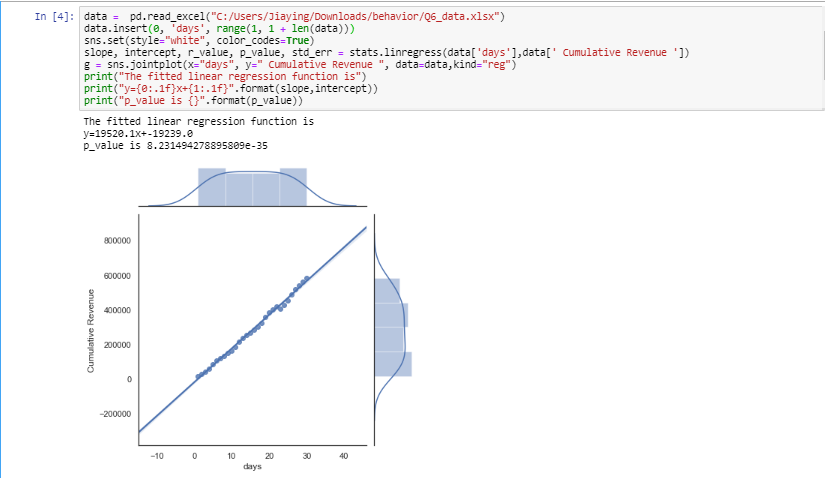
6.

The answer is 4th Feb.

First, I made a quick plot in excel (click on insert, choose line plot), it gives a scatter plot as **Fig.7**. We can observe an almost linear relationship between date and cumulative revenue, it seems we can reach 700k pretty soon following this pattern.

**Fig.7 Scatter plot made by excel.**

To confirm my guess, I used “stats. linregress” function in scipy python package (**Fig.8**), it confirms the linearity with p value 8.23e-35. According to the linear function it gives us: y = 19520x – 19239, we will reach 700k after 34 days since the first day 2019-01-01, which leads to 4th Feb, the cumulative revenue will reach 700k.



**Fig.8 Prediction to reach 700k in python.**