SevenApps Data Analysis Task Solutions

Uğurcan Demir

18/08/2025

Part 1: SQL Implementation

In order to do the funnel analysis a dummy dataset was generated and uploaded to the locally deployed PostgreSQL database. The Python script that generated the dataset and uploaded it to the database can be found in this folder with name **generate_and_upload_data.py**. The data was generated strictly in accordance with the given instructions and it is also stored in this folder with name **user_events.csv**.

The following SQL query was written on a PostgreSQL system but can be used in any major RDBMS either without any change or with a few minor changes. The whole quesy and its output are also stored in this folder of task solutions, with names **funnel.sql** and **funnel_output.csv**.

We begin our analysis by creating a starter table only with necessary columns.

```
WITH ordered_events AS (
    SELECT
        user_id,
        platform,
        event_name,
        timestamp
    FROM user_events
),
```

We proceed by detecting the "'PageView'" cases and checking if , for each user, they are followed by other actions such as "Download", "Install", and "Purchase", and within 72 hours. This method incrementally enlarges our starter table.

```
-- Step 1: PageView
pageviews AS (
    SELECT user_id, platform, MIN(timestamp) AS pageview_time
    FROM ordered_events
    WHERE event_name = 'PageView'
    GROUP BY user_id, platform
),

-- Step 2: Download within 72h of PageView
downloads AS (
    SELECT p.user_id, p.platform, MIN(d.timestamp) AS download_time
    FROM pageviews p
    JOIN ordered_events d
      ON d.user_id = p.user_id
```

```sql
     AND d.platform = p.platform
     AND d.event_name = 'Download'
     AND d.timestamp > p.pageview_time
     AND d.timestamp <= p.pageview_time + INTERVAL '72 hours'
    GROUP BY p.user_id, p.platform
),

-- Step 3: Install within 72h of Download
installs AS (
    SELECT d.user_id, d.platform, MIN(i.timestamp) AS install_time
    FROM downloads d
    JOIN ordered_events i
      ON i.user_id = d.user_id
     AND i.platform = d.platform
     AND i.event_name = 'Install'
     AND i.timestamp > d.download_time
     AND i.timestamp <= d.download_time + INTERVAL '72 hours'
    GROUP BY d.user_id, d.platform
),

-- Step 4: Purchase within 72h of Install
purchases AS (
    SELECT i.user_id, i.platform, MIN(pu.timestamp) AS purchase_time
    FROM installs i
    JOIN ordered_events pu
      ON pu.user_id = i.user_id
     AND pu.platform = i.platform
     AND pu.event_name = 'Purchase'
     AND pu.timestamp > i.install_time
     AND pu.timestamp <= i.install_time + INTERVAL '72 hours'
    GROUP BY i.user_id, i.platform
),

-- Funnel: combine all steps
funnel AS (
    SELECT
        p.user_id,
        p.platform,
        p.pageview_time,
        d.download_time,
        i.install_time,
        pu.purchase_time
    FROM pageviews p
    LEFT JOIN downloads d ON d.user_id = p.user_id AND d.platform = p.platform
    LEFT JOIN installs i  ON i.user_id = p.user_id AND i.platform = p.platform
    LEFT JOIN purchases pu ON pu.user_id = p.user_id AND pu.platform =
p.platform
)
```

In “funnel” table we have combined all the necessary information to draw conclusions. What we did so far basically is pivoting **user_events** to a wider table by breaking down the **timestamp** column for each user with **72 hours** condition.

Now we are ready for the computations. The part of query below, calculates how strongly users went through from viewing the page and purchasing the product, **both in terms of absolute numbers and conversion rates broken down by platform**, just as requested.

```
-- Final aggregation with counts + conversion rates
SELECT
    platform,
    COUNT(DISTINCT CASE WHEN pageview_time IS NOT NULL THEN user_id END) AS
pageviews,
    COUNT(DISTINCT CASE WHEN download_time IS NOT NULL THEN user_id END) AS
downloads,
    COUNT(DISTINCT CASE WHEN install_time IS NOT NULL THEN user_id END) AS
installs,
    COUNT(DISTINCT CASE WHEN purchase_time IS NOT NULL THEN user_id END) AS
purchases,

    ROUND(
        CAST(COUNT(DISTINCT CASE WHEN download_time IS NOT NULL THEN user_id
END) AS DECIMAL(10,3))
        / NULLIF(COUNT(DISTINCT CASE WHEN pageview_time IS NOT NULL THEN
user_id END), 0),
        3
    ) AS cv_pageview_to_download,

    ROUND(
        CAST(COUNT(DISTINCT CASE WHEN install_time IS NOT NULL THEN user_id
END) AS DECIMAL(10,3))
        / NULLIF(COUNT(DISTINCT CASE WHEN download_time IS NOT NULL THEN
user_id END), 0),
        3
    ) AS cv_download_to_install,

    ROUND(
        CAST(COUNT(DISTINCT CASE WHEN purchase_time IS NOT NULL THEN user_id
END) AS DECIMAL(10,3))
        / NULLIF(COUNT(DISTINCT CASE WHEN install_time IS NOT NULL THEN
user_id END), 0),
        3
    ) AS cv_install_to_purchase

FROM funnel
GROUP BY platform
ORDER BY platform;
```

The result of the funnel analysis can be found below. We can clearly see that, based on our (fabricated) data, android users constitute a slightly larger portion of our user base. However, ios users are more likely to purchase a product after having installed it which may be a sign of profitability.

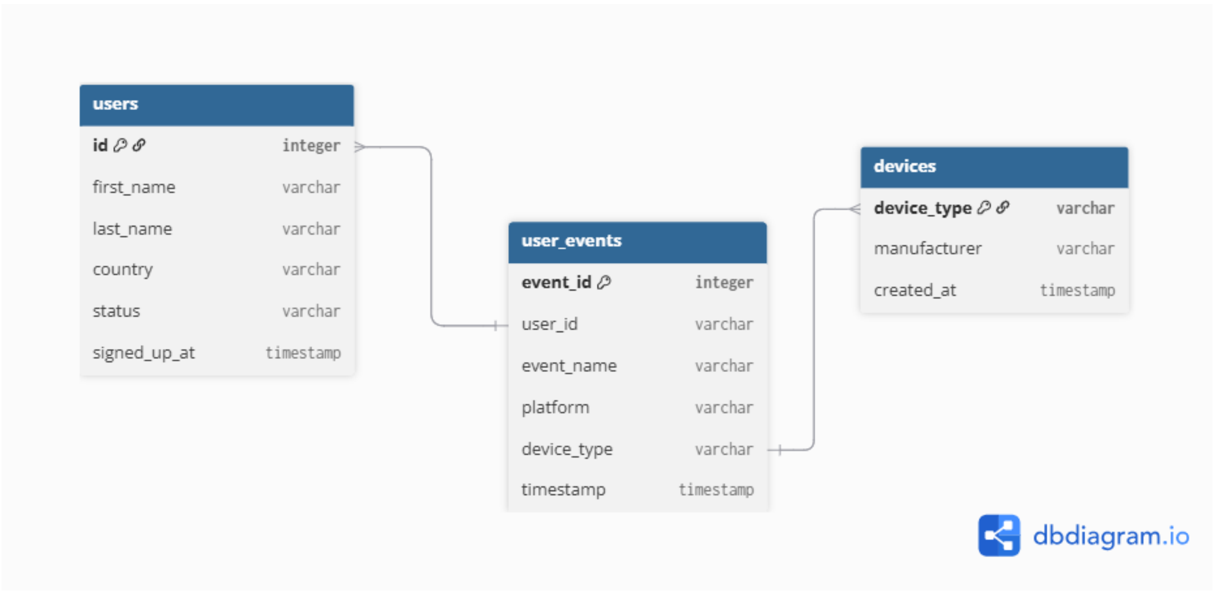
platform	pageviews	downloads	installs	purchases	cv_pageview	cv_downloa	cv_install_to_p
android	2713	1341	632	118	0.494	0.471	0.187
ios	2664	1278	562	117	0.48	0.44	0.208

Part 2: Data Modeling Questions

Q1 & Q2)

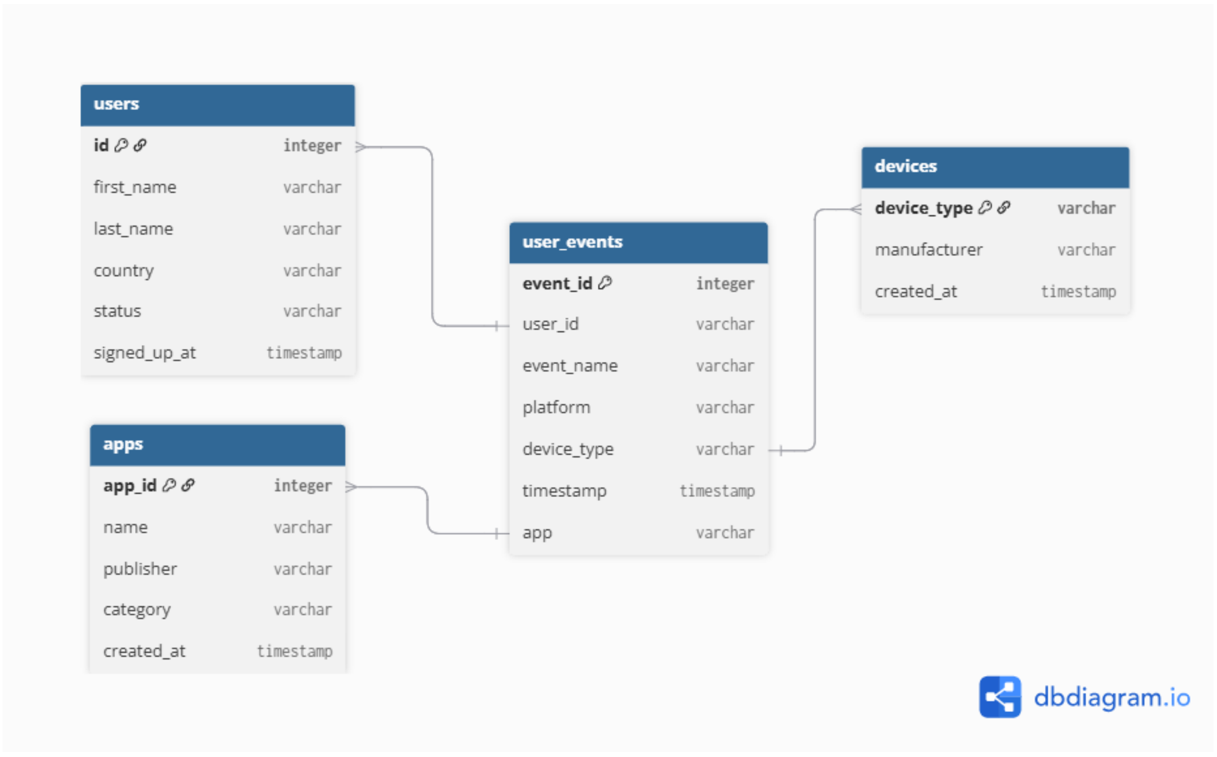
This application seems like a perfect subject for **Star schema** (a special case of **Snowflake schema**) data model. I would treat the **user_events** table as the fact table and treat other columns as foreign keys of dimension tables. This would help a lot when it comes to partitioning also immensely help when we wish to add attributes like country and device type.

Below, we show a simple example of a data model that would suit this process. This is the data model I would use in production. It can be clearly seen that it helps with a clear table structure , partitioning and easily adding new columns like country and device type.



Q3)

The problem with the current event tracking system is that it doesn’t record that app/product these event are happening on. In other word it doesn’t tell upon what, the users are viewing installing purchasing etc. In order to overcome that obstacle, we can advance our data model as below. Because it would improve our current event tracking system.



Part 3: Visualization

Q1)

There are many options to go. One can use the utilities of programming languages like Python and R. BI tools are also good options for visualization and dashboard design.

- **Python**:
  - matplotlib / seaborn / plotly : quick and interactive plots
  - Streamlit : good for creating dashboards like web apps.
- **R**:
  - ggplot2 : R's best library.
  - shiny : equivalent of straemlit in R ecosystem.
- **BI tools** :
  - **Tableau**/ **Power BI** / **Looker** : well known BI apps
  - **Metabase** / **Apache Superset** : open-source alternatives.
- **Special mention**: If this dataset goes into a data warehouse (like BigQuery or Redshift), you could plug **dbt + BI tool** for scalable monitoring.

Q2)

Dashboard design to monitor conversion rates:

- **Funnel chart** showing the number of users at each step: PageView → Download → Install → Purchase.
- **Line chart over time** to track how conversion rates change daily or weekly.
- **Filters** to break down by platform (iOS/Android) or device type (phone/tablet).
- Optional: highlight drop-offs between steps to see where most users leave the funnel.

Q3)

To make the analysis richer, I'd include:

- **Drop-off rates** (where users are leaking from the funnel).
- **Average time between steps**:
  - How long on average it takes users to go from Download → Install, Install → Purchase.
- **Breakdowns**:
  - **By platform** (iOS vs Android).
  - **By device type** (phone vs tablet).
  - **By time** (day/week/month trends).
  - Possibly **by user cohorts** (first activity date).
- **Anomalies**:
  - Sudden spikes/drops in conversion (alerts).