**SQL Task Solution (Vladislavs Nanaks)**

SQL Task

You have two very large tables.

‘Audience’. This table contains one row per play event, and is partitioned by day.

‘Demographic’. This table contains demographic informations about our accounts.

Please write queries to answer the following two questions:

1. What is the average completion rate per content\_type?
2. What is the average play time by age group (25-34,35-44,45-55,55+)?

Introduction

When reviewing the task, I noticed the statement regarding the two tables being very large. Therefore, I decided to focus my solution on optimising query performance, assuming that the final dataset will be used frequently.

This task was solved using MySQL and MySQL Workbench, so the syntax might differ from other SQL types.

In summary, the two provided tables were combined into a larger, more comprehensive table that is refreshed once per day. This larger table was then aggregated into a smaller table with additional metrics for insight gathering. The smaller aggregate table also has a refresh interval of one day. The aggregated table includes more data than is strictly necessary to answer the provided questions. The reason for this approach is the assumption that the dataset will be used frequently and will serve as the foundation for a complete user engagement dashboard, providing a wide range of insights. I aimed to create a table that would be highly flexible in its use and application while still improving performance compared to querying the initial two tables directly.

Below, alongside the code, you will find a description of the decisions made and the reasoning behind them.

Additionally, I have outlined the limitations of the current solution as well as recommendations for future improvements and code refinement.

Finally, in the appendix, you will find the supporting code used to create the task tables.

Data Preprocessing Code

**-- DERIVATIVE TABLE CREATION**

First, I created a derivative table that combines the two original tables. While this approach requires more storage space, it eliminates the need for repeated join queries and reduces query complexity.

CREATE TABLE IF NOT EXISTS combined\_demographic\_audience\_view AS

SELECT

audience.event\_id,

audience.user\_id,

audience.date AS date\_timestamp,

audience.episode\_id,

audience.content\_type,

audience.play\_time,

audience.episode\_length,

demographic.age,

demographic.region

FROM

audience

LEFT JOIN

demographic

ON

audience.user\_id = demographic.user\_id;

Aggregate Table Creation: This is one of the main steps in the solution. Currently, the table is aggregated by date to accommodate instances where users engage with the content more than once per day. This approach provides an example of the methodology, but the table can be further optimised. For example, if the episode\_id is not critical to stakeholders, it can be removed, which will subsequently increase the watch\_instance\_count, play\_length, and other metrics.

Another way to reduce the table size is by aggregating the date further (e.g., by week or month). However, in this case, the daily granularity was retained to preserve the timeline of engagement for future analysis.

This approach also simplifies data analysis by providing more consistent results compared to real-time tables, as the table is refreshed once a day.

With this table, stakeholders can track activity trends and user engagement, as well as monitor these trends over time.

COALESCE clauses were added to handle null values. However, such instances indicate an issue where the audience table includes user\_ids that are not present in the demographic table (in this case, the reference table).

The addition of individual completion\_rate\_percentage calculations was performed at this level intentionally to improve performance compared to writing queries directly on top of the raw table.

CREATE TABLE IF NOT EXISTS user\_engagement\_aggregate AS

SELECT

user\_id,

DATE(date\_timestamp) AS date,

episode\_id,

content\_type,

episode\_length,

COALESCE(age, 'Unknown') as user\_age,

COALESCE(region, 'Unknown') as user\_region,

COUNT(\*) AS watch\_instance\_count,

SUM(play\_time) AS total\_play\_time,

ROUND(SUM(play\_time) / episode\_length \* 100, 2) AS completion\_rate\_percentage

FROM

combined\_demographic\_audience\_view

GROUP BY

user\_id, DATE(date\_timestamp), episode\_id, content\_type, age, region, episode\_length;

Example Data:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **user\_id** | **date** | **episode\_id** | **content\_ type** | **episode\_ length** | **user\_age** | **user\_region** | **watch\_ instance\_ count** | **total\_ play\_ time** | **completion\_ rate\_ percentage** |
| xw8EF | 8/25/2016 | woaqa01 | vod | 200 | 25 | North West | 1 | 180 | 90 |
| KuFh7 | 8/25/2016 | UddhdS01 | live | 1100 | Unknown | Unknown | 1 | 1000 | 90.91 |
| IfTI1 | 8/25/2016 | eodjdm02 | vod | 300 | 30 | London | 1 | 260 | 86.67 |
| Li9IS | 8/25/2016 | wneola02 | live | 1450 | 40 | Scotland | 1 | 80 | 5.52 |
| LiF7q | 8/25/2016 | endoep03 | live | 720 | 51 | North Wales | 1 | 700 | 97.22 |
| LhTcE | 8/25/2016 | Wjdjdkdk02 | vod | 300 | 60 | London | 1 | 190 | 63.33 |
| LiKkg | 8/25/2016 | ddhdk02 | vod | 500 | 45 | South West | 1 | 360 | 72 |
| Kokho | 8/25/2016 | oensksl02 | vod | 400 | Unknown | Unknown | 1 | 389 | 97.25 |

**-- TABLE UDATE**

This section explains how the tables are refreshed. A refresh EVENT was created to execute the truncation of both tables and the insertion of up-to-date results from the two initial tables. This process is scheduled to run once a day.

DELIMITER //

CREATE EVENT combined\_demographic\_audience\_view\_refresh

ON SCHEDULE EVERY 1 DAY

DO

BEGIN

TRUNCATE TABLE combined\_demographic\_audience\_view;

INSERT INTO combined\_demographic\_audience\_view

SELECT

audience.event\_id,

audience.user\_id,

audience.date AS date\_timestamp,

audience.episode\_id,

audience.content\_type,

audience.play\_time,

audience.episode\_length,

demographic.age,

demographic.region

FROM

audience

LEFT JOIN

demographic

ON

audience.user\_id = demographic.user\_id;

END//

DELIMITER ;

DELIMITER //

CREATE EVENT user\_engagement\_aggregate\_refresh

ON SCHEDULE EVERY 1 day

DO

BEGIN

TRUNCATE TABLE user\_engagement\_aggregate;

INSERT INTO user\_engagement\_aggregate

SELECT

user\_id,

DATE(date\_timestamp) AS date,

episode\_id,

content\_type,

episode\_length,

COALESCE(age, 'Unknown') as age,

COALESCE(region, 'Unknown') as region,

COUNT(\*) AS watch\_instance\_count,

SUM(play\_time) AS total\_play\_time,

ROUND(SUM(play\_time) / episode\_length \* 100, 2) AS completion\_rate\_percentage

FROM

combined\_demographic\_audience\_view

GROUP BY

user\_id, DATE(date\_timestamp), episode\_id, content\_type, age, region, episode\_length;

END//

DELIMITER ;

Answering questions

1. What is the average completion rate per content\_type?

SELECT

content\_type,

ROUND(AVG(total\_play\_time), 2) AS avg\_play\_time,

ROUND(AVG(episode\_length), 2) AS avg\_episode\_length,

ROUND(AVG(completion\_rate\_percentage), 2) AS avg\_completion\_rate\_percentage

FROM

user\_engagement\_aggregate

GROUP BY

content\_type;

|  |  |  |  |
| --- | --- | --- | --- |
| **content\_type** | **avg\_play\_time** | **avg\_episode\_length** | **avg\_completion\_rate\_percentage** |
| vod | 275.8 | 340 | **81.85** |
| live | 593.33 | 1090 | **64.55** |

1. What is the average play time by age group (25-34,35-44,45-55,55+)?

SELECT

CASE

WHEN user\_age = 'Unknown' THEN 'Unknown'

WHEN user\_age BETWEEN 25 AND 34 THEN '25-34'

WHEN user\_age BETWEEN 35 AND 44 THEN '35-44'

WHEN user\_age BETWEEN 45 AND 54 THEN '45-54'

WHEN user\_age < 25 THEN 'Less Than 25'

ELSE '55+'

END AS age\_group,

ROUND(AVG(total\_play\_time), 2) AS avg\_play\_time\_in\_minutes

FROM user\_engagement\_aggregate

GROUP BY age\_group

ORDER BY age\_group ASC;

|  |  |
| --- | --- |
| **age\_group** | **avg\_play\_time\_in\_minutes** |
| 25-34 | 220 |
| 35-44 | 80 |
| 45-54 | 530 |
| 55+ | 190 |
| Unknown | 694.5 |

Limitations and Future Refinement

* Takes more space.   
  This solution has several limitations that should be addressed before implementing it. Firstly, creating a separate joined table will require double the amount of space compared to using just the two separate tables.  
  + The necessity of having such a detailed, line-level table should be assessed, as the code can be adapted to only include the aggregate table.
* Table Truncation  
  Truncating the table and reinserting values is not the most efficient method for refreshing the table.  
  + For future refinement, it would be beneficial to explore alternative methods for updating the tables, such as dropping and recreating the table or using materialised views.
* Issue with episode rewatching.   
  There is a potential issue where metrics like the average completion rate could be skewed if users rewatch certain episodes, resulting in engagement time exceeding the episode length.  
  + Further investigation is required to understand and address this issue.

Further recommendations:

* Data Testing/Investigation  
  I would strongly recommend investigating the data further and performing normalisation to address inconsistencies, issues, and outliers that might impact data integrity and quality.
* Engagement During the Day

Although not part of the current task, I believe stakeholders might benefit from understanding when, and at what time of day, users and specific age groups engage with the content. To achieve this, I would extract the hour of row creation from the timestamp into a separate column and include it in the aggregate table.

* Table Refresh Time

I suggest setting the refresh time for the combined table to 23:59 and the aggregate table to 00:01 the following day. This approach would eliminate incomplete results for the current day when performing analysis. Stakeholders are likely to benefit more from having full data up to the previous day rather than partial results from the current day.

Appendix

**-- DATABASE CREATION AND USE**

CREATE DATABASE BBC;

USE BBC;

**-- INITIAL TABLE CREATION AND DATA INPUT**

CREATE TABLE IF NOT EXISTS audience (

event\_id CHAR(16) NOT NULL, -- Fixed-length unique event identifier

user\_id CHAR(6) NOT NULL, -- Short user ID

date TIMESTAMP NOT NULL, -- Timestamp for the event

episode\_id CHAR(10) NOT NULL, -- Fixed-length episode identifier

content\_type ENUM('vod', 'live') NOT NULL, -- Limited to 'vod' and 'live'

play\_time SMALLINT UNSIGNED NOT NULL, -- Play time in seconds (up to ~65,535)

episode\_length SMALLINT UNSIGNED NOT NULL, -- Episode length in seconds

PRIMARY KEY (event\_id)

);

INSERT INTO audience (event\_id, user\_id, date, episode\_id, content\_type, play\_time, episode\_length) VALUES

('iqeq4x84r6x2yxl', 'xw8EF', '2016-08-25 08:15:30', 'woaqa01', 'vod', 180, 200),

('iqeujzl76pff8hu', 'IfTI1', '2016-08-25 08:16:31', 'eodjdm02', 'vod', 260, 300),

('iqfnedekranvo17', 'LiKkg', '2016-08-25 08:15:35', 'ddhdk02', 'vod', 360, 500),

('iqf9z35v5qxhsky', 'LiF7q', '2016-08-25 08:15:39', 'endoep03', 'live', 700, 720),

('iqf6dth7il4ta0m', 'Li9IS', '2016-08-25 08:15:30', 'wneola02', 'live', 80, 1450),

('iqfn8x5urtl1s98', 'LhTcE', '2016-08-25 08:17:30', 'Wjdjdkdk02', 'vod', 190, 300),

('iqerdwmt2u3bb1n', 'KuFh7', '2016-08-25 08:17:43', 'UddhdS01', 'live', 1000, 1100),

('iqg0u88i82wlr1i', 'Kokho', '2016-08-25 08:15:54', 'oensksl02', 'vod', 389, 400);

CREATE TABLE IF NOT EXISTS demographic (

age TINYINT UNSIGNED NOT NULL,

user\_id CHAR(6) NOT NULL,

region VARCHAR(50) NOT NULL,

PRIMARY KEY (user\_id));

INSERT INTO demographic (age, user\_id, region)

VALUES

(25, 'xw8EF', 'North West'),

(30, 'IfTI1', 'London'),

(45, 'LiKkg', 'South West'),

(51, 'LiF7q', 'North Wales'),

(40, 'Li9IS', 'Scotland'),

(60, 'LhTcE', 'London'),

(47, 'GgeoR', 'North East');