

GROUP 5

DELIVERABLE 4 (Part 1)

1) View with CTE and Functions (Ranking, Partition By and Round) with Sorting

Name: vw_latestweight

Output: Table with device_id, date, metric_name, weight, rounded_bmi

Use Case: The user logs his weight on the mobile application. This view shows the latest weight for each device_id with the date when it was logged on, the metric they were recorded in and the corresponding bmi rounded off to two decimals.

Script:

```
CREATE VIEW vw_latestweight AS
WITH weight_and_metric AS (
    SELECT device_id,
           date,
           metric_name,
           Round(weightpounds,2) AS weight,
           Round(bmi,2) as rounded_bmi,
           ROW_NUMBER() OVER(PARTITION BY device_id ORDER BY `date` DESC) AS
row_num
    FROM
        weightloginfo, weight_metric
    WHERE
        weightloginfo.metric_id = weight_metric.id
)
SELECT device_id,
       `date`,
       metric_name,
       weight,
       rounded_bmi
FROM weight_and_metric
WHERE row_num = 1;
```

Results:

```
1 • CREATE VIEW vw_latestweight AS
2 WITH weight_and_metric AS (
3     SELECT device_id,
4             date,
5             metric_name,
6             Round(weightpounds,2) AS weight,
7             Round(bmi,2) as rounded_bmi,
8             ROW_NUMBER() OVER(PARTITION BY device_id ORDER BY `date` DESC) AS row_num
9     FROM
10        weightloginfo, weight_metric
11
12     WHERE
13        weightloginfo.metric_id = weight_metric.id
14 )
15 SELECT device_id,
16        `date`
```

Output

Action Output

#	Time	Action	Message
✓ 1	17:21:31	CREATE VIEW vw_latestweight AS WITH weight_and_metric AS (SELE...	0 row(s) affected

SELECT Query

*SELECT * FROM vw_latestweight;*

```
1 • SELECT * FROM vw_latestweight;
```

device_id	date	metric_name	weight	rounded_bmi
1503960366	2016-05-03 23:59:59	Pounds	115.96	22.65
1927972279	2016-04-13 01:08:52	Pounds	294.32	47.54
2873212765	2016-05-12 23:59:59	Pounds	126.32	21.69
4319703577	2016-05-04 23:59:59	Pounds	159.39	27.38
4558609924	2016-05-09 23:59:59	Pounds	152.34	27
5577150313	2016-04-17 09:17:55	Pounds	199.96	28
6962181067	2016-05-12 23:59:59	Pounds	136.47	24.17
8877689391	2016-05-12 06:42:53	Pounds	185.19	25.14

2) View with Aggregate Functions, Conditions (IF clause) Statement, ROLL UP (Summary Function), Group By and Sorting:

Name: vw_totalCaloriesByMonth

Output: Table with device_id, month, total_calories

Use Case: The device records calories burned by the wearer every minute. This view returns the total number of calories burned monthly for each device id.

Script:

```
SET GLOBAL sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY',''));
```

```
CREATE VIEW vw_totalCaloriesByMonth AS
SELECT
    device_id,
    IF(GROUPING(MONTHNAME(activityminute)), 'Total',
MONTHNAME(activityminute)) AS month,
    SUM(calories) AS Total_Calories
FROM calorie_mins
GROUP BY device_id, MONTHNAME(activityminute) WITH ROLLUP
ORDER BY device_id, MONTHNAME(activityminute)
```

Results:



The screenshot shows a SQL IDE with a script editor and an output pane. The script editor contains the following SQL code:

```
1 • CREATE VIEW vw_totalCaloriesByMonth AS
2     SELECT
3         device_id,
4         IF(GROUPING(MONTHNAME(activityminute)), 'Total', MONTHNAME(activityminute)) AS month,
5         SUM(calories) AS Total_Calories
6     FROM calorie_mins
7     GROUP BY device_id, MONTHNAME(activityminute) WITH ROLLUP
8     ORDER BY device_id, MONTHNAME(activityminute)
9
```

The output pane shows the execution results:

#	Time	Action	Message
1	17:27:06	CREATE VIEW vw_totalCaloriesByMonth AS SELECT device_id, IF(GRO...	0 row(s) affected

Select Query:

```
SELECT * FROM vw_totalCaloriesByMonth;
```

1 • `SELECT * FROM vw_totalCaloriesByMonth;`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	device_id	month	Total_Calories
▶	NULL	Total	2073239
	1503960366	Total	61425
	1503960366	April	39056
	1503960366	May	22369
	1624580081	Total	51944
	1624580081	April	32042
	1624580081	May	19902
	1644430081	Total	75373
	1644430081	April	48864
	1644430081	May	26509
	1844505072	Total	50203
	1844505072	April	32856
	1844505072	May	17347
	1927972279	Total	51134
	1927972279	April	31643
	1927972279	May	19491
	2022484408	Total	75104
	2022484408	April	47140

vw_totalCaloriesByMonth 3 ×

3) View with Nested/Sub queries, Joins and Aggregate Functions:

Name: vw_activity_summary

Output: Table with fitbit_id, activity_name, total_calories_burned, total_active_minutes and date

Use Case: The user logs his activities with intensities and the number of minutes spent doing the activity at the given intensity level. This view shows the summary with total minutes and calories burned by the user doing an activity each day.

For E.g., a user may do cycling at low intensity for warm up for 10 mins and 5 mins to cool down and at high intensity for 15 mins. There are different number of calories burned for different activities based on intensity levels. In this user's case, the view will show total of 30 mins spent on cycling with total calories as (calories at low intensities * (10+5 minutes) + calories at high intensity * 15 mins)

Script:

```
CREATE VIEW vw_activity_summary AS
SELECT
    al.fitbit_id,
    am.activity_name,
    SUM(al.minutes * am.calories_burned_per_minute) AS total_calories_burned,
    SUM(al.minutes) AS total_active_minutes,
    DATE(al.timestamp) AS `date`
FROM (SELECT * FROM activity_log_daily
      WHERE fitbit_id IN (SELECT DISTINCT device_id from steps_mins WHERE steps > 0)
    ) AS al
```

```

INNER JOIN activity_master AS am ON al.activity_id = am.activity_id AND al.intensity =
am.intensity
GROUP BY al.fitbit_id, am.activity_name, DATE(al.timestamp)

```

Results:

The screenshot shows a SQL IDE interface. The top pane contains a SQL script to create a view named `vw_activity_summary`. The script uses a subquery to select data from `activity_log_daily` based on a filter from `steps_mins`, then joins it with `activity_master` and groups the results by `fitbit_id`, `activity_name`, and `DATE(timestamp)`.

```

1 • CREATE VIEW vw_activity_summary AS
2   SELECT
3     al.fitbit_id,
4     am.activity_name,
5     SUM(al.minutes * am.calories_burned_per_minute) AS total_calories_burned,
6     SUM(al.minutes) AS total_active_minutes,
7     DATE(al.timestamp) AS `date`
8   FROM (SELECT * FROM activity_log_daily
9         WHERE fitbit_id IN (SELECT DISTINCT device_id from steps_mins WHERE steps > 0)
10        ) AS al
11  INNER JOIN activity_master AS am ON al.activity_id = am.activity_id AND al.intensity = am.intensity
12  GROUP BY al.fitbit_id, am.activity_name, DATE(al.timestamp)

```

The bottom pane shows the 'Output' tab with 'Action Output' selected. It displays a single message indicating the successful execution of the SQL script.

#	Time	Action	Message
1	17:14:36	CREATE VIEW vw_activity_summary AS SELECT al.fitbit_id, am.activit...	0 row(s) affected

Select Query:

```
SELECT * FROM vw_activity_summary;
```

The screenshot shows a SQL IDE interface. The top pane contains a SQL query to select all data from the `vw_activity_summary` view.

```

1 • SELECT * FROM vw_activity_summary;

```

The bottom pane shows the 'Result Grid' tab. It displays a table with 6 columns: `fitbit_id`, `activity_name`, `total_calories_burned`, `total_active_minutes`, and `date`. The table contains 15 rows of data.

fitbit_id	activity_name	total_calories_burned	total_active_minutes	date
1503960366	kickboxing	1800	30	2016-04-12
1503960366	dance	1650	30	2016-04-12
1624580081	cycling	300	15	2016-04-12
1624580081	aerobics	2025	45	2016-04-12
1644430081	cycling	300	15	2016-04-12
1644430081	running	675	45	2016-04-12
1844505072	yoga	300	30	2016-04-12
1844505072	pilates	720	60	2016-04-12
1927972279	weightlifting	3675	75	2016-04-12
2022484408	kickboxing	1500	25	2016-04-12
2022484408	dance	1650	30	2016-04-12

At the bottom of the result grid, there is a tab labeled `vw_activity_summary 15` with a close button.