1. Trigger: UpdateChannelViews

```
CREATE DEFINER=`root`@`%` TRIGGER `UpdateChannelViews` AFTER INSERT ON
`Video` FOR EACH ROW BEGIN
   IF NOT EXISTS (SELECT 1 FROM Channel WHERE ChannelId = NEW.ChannelId)
THEN
      INSERT INTO Channel VALUES (NEW.ChannelId, NEW.ChannelName, 0);
   END IF;

   UPDATE Channel
   SET TotalViews = TotalViews + NEW.Views
   WHERE Channel.ChannelId = NEW.ChannelId;
END
```

The UpdateChannelViews trigger is defined to execute after a new record is inserted into the Video table. The trigger checks if the channel associated with the new video exists in the Channel table, and if it doesn't, it creates a new channel record with an initial total views count of 0. After this check, the trigger updates the TotalViews field for the corresponding channel in the Channel table, adding the number of views of the newly inserted video. This trigger provides useful functionality by automatically maintaining an accurate count of total views for each channel, ensuring the application's data remains consistent and up-to-date without requiring manual updates or additional queries.

FYI, we created two stored procedures that each includes one advanced query. They sum up to 2 in total!

2. First Stored Procedure: GetPopularVideosByCategory

```
CREATE DEFINER=`root`@`%` PROCEDURE `GetPopularVideosByCategory`(IN
categoryId INT)
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE videoId VARCHAR(255);
   DECLARE videoTitle VARCHAR(255);
   DECLARE channelTitle VARCHAR(255);
   DECLARE views INT;
   DECLARE channelId INT;
   DECLARE avgViews INT;

   DECLARE cur CURSOR FOR
      SELECT v.VideoId, v.VideoTitle, ch.ChannelTitle, v.Views, ch.ChannelId
      FROM Video v
      JOIN Channel ch ON v.ChannelId = ch.ChannelId
      JOIN Category c ON ch.CategoryId = c.CategoryId
      WHERE v.CategoryId = categoryId AND v.Views > 100000
      AND v.VideoId IN (
         SELECT sub.VideoId
         FROM (
            SELECT v.VideoId, RANK() OVER (PARTITION BY ch.ChannelId ORDER BY
v.Views DESC) AS rank
            FROM Video v
            JOIN Channel ch ON v.ChannelId = ch.ChannelId
            JOIN Category c ON ch.CategoryId = c.CategoryId
            WHERE v.CategoryId = categoryId AND v.Views > 100000
         ) sub
         WHERE sub.rank <= 10
      )
      ORDER BY v.Views DESC;

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

   DROP TABLE IF EXISTS `PopularVideos`;
```

```
    CREATE TABLE PopularVideos (
        VideoId VARCHAR(255) PRIMARY KEY,
        VideoTitle VARCHAR(255),
        ChannelTitle VARCHAR(255),
        Views INT
    );

    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO videoId, videoTitle, channelTitle, views, channelId;

        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Calculate the average views for the given channel
        SELECT AVG(v.Views) INTO avgViews
        FROM Video v
        WHERE v.ChannelId = channelId;

        -- If the video's views are more than the channel's average views, insert into the
PopularVideos table
        IF views > avgViews THEN
            INSERT INTO PopularVideos (VideoId, VideoTitle, ChannelTitle, Views)
            VALUES (videoId, videoTitle, channelTitle, views);
        END IF;
    END LOOP;

    CLOSE cur;

    SELECT * FROM PopularVideos;
END;
```

The stored procedure GetPopularVideosByCategory takes a categoryId as input and returns popular videos in that category based on two criteria: the video must have over 100,000 views and its views must be greater than the average views of the channel it belongs to. It employs advanced SQL concepts such as subqueries, join of multiple relations, and aggregation, as well as cursors, loops, and control structures to fetch the top 10 videos per channel in the specified category, calculate each channel's average views, and filter the results to only include videos with views above their channel's average. The resulting data is inserted into a table called PopularVideos, which is then returned as the final output.

This stored procedure is useful for this application as it allows users of the database to identify the popularity and engagement of certain categories of videos. This could be useful to people interested in knowing which categories are trending and which may consistently generate the most user engagement.

3. Second Stored Procedure: GetTopChannelsByAverageVideoViews

```
CREATE PROCEDURE GetTopChannelsByAverageVideoViews(IN topN INT)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur_ChannelId VARCHAR(255);
    DECLARE cur_ChannelTitle VARCHAR(255);
    DECLARE cur_AvgViews FLOAT;
    DECLARE cur_TopNVideoCount INT;

    DECLARE channel_cursor CURSOR FOR
        SELECT ch.ChannelId, ch.ChannelTitle, AVG(v.Views) AS AvgViews,
COUNT(topNVideos.VideoId) AS TopNVideoCount
        FROM Channel ch
        JOIN Video v ON ch.ChannelId = v.ChannelId
        LEFT JOIN (
            SELECT VideoId, ChannelId
            FROM Video
            WHERE Views >= (
                SELECT Views
                FROM Video
                ORDER BY Views DESC
                LIMIT 1 OFFSET (topN - 1)
            )
        ) topNVideos ON ch.ChannelId = topNVideos.ChannelId
        GROUP BY ch.ChannelId, ch.ChannelTitle
        ORDER BY AvgViews DESC;

    -- Handler for cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Create a temporary table to store the results
    DROP TABLE IF EXISTS TopChannels;
    CREATE TABLE TopChannels (
        ChannelId VARCHAR(255),
        ChannelTitle VARCHAR(255),
        AvgViews FLOAT,
        TopNVideoCount INT
    );
```

```
   OPEN channel_cursor;

   FETCH channel_cursor INTO cur_ChannelId, cur_ChannelTitle, cur_AvgViews,
cur_TopNVideoCount;

   my_loop: LOOP
      IF done THEN
         LEAVE my_loop;
      END IF;

      INSERT INTO TopChannels (ChannelId, ChannelTitle, AvgViews,
TopNVideoCount) VALUES (cur_ChannelId, cur_ChannelTitle, cur_AvgViews,
cur_TopNVideoCount);

      FETCH channel_cursor INTO cur_ChannelId, cur_ChannelTitle, cur_AvgViews,
cur_TopNVideoCount;
   END LOOP;

   CLOSE channel_cursor;

   SELECT * FROM TopChannels;
END
```

The stored procedure GetTopChannelsByAverageVideoViews takes an input parameter topN and returns the top channels with the highest average views per video and the total number of videos in the topN percentile by views. It employs advanced SQL concepts such as subqueries and aggregation, as well as cursors, loops, and control structures to calculate the average views for each channel, retrieve the top topN videos by views, and count the number of these top videos per channel. The resulting data is inserted into a table called TopChannels, which is then returned as the final output.

This stored procedure can provide valuable insights into the most popular channels by identifying those with the highest average video views, helping users discover content that resonates with a larger audience. Additionally, it enables content creators and marketers to analyze the top-performing channels to gain inspiration and better understand the factors driving success on the platform.