```
USE BlogApplication;
SELECT * FROM BlogUsers;
SELECT * FROM BlogPosts;
SELECT * FROM BlogComments;
SELECT * FROM Reshares;


# 1. Business question: do we have data that doesn't make sense?
# Are there any unpublished posts that have comments? This checks for unexpected data.
SELECT BlogPosts.PostId, BlogPosts.Title, BlogPosts.UserName AS BLOG_USER,
  BlogComments.Created, BlogComments.UserName AS COMMENT_USER
FROM BlogPosts INNER JOIN BlogComments
  ON BlogPosts.PostId = BlogComments.PostId
WHERE BlogPosts.Published = False;
# SELECT * FROM BlogPosts INNER JOIN BlogComments ON BlogPosts.PostId = BlogComments.PostId;


# 2. Most recent comment for each UserName.
SELECT UserName, MAX(Created)
FROM BlogComments
GROUP BY UserName;
# SELECT * FROM BlogComments;


# 3. Count of each UserName's comments.
SELECT UserName, COUNT(*) AS CNT
FROM BlogComments
GROUP BY UserName
ORDER BY CNT DESC, UserName DESC;
# LIMIT 2;  # Max rows to return.
# LIMIT 2 OFFSET 1;  # With an offset.


# 4. UserNames who commented more than once.
SELECT UserName, COUNT(*) AS CNT
FROM BlogComments
GROUP BY UserName
HAVING COUNT(*) > 1
#HAVING CNT > 1  # Non-standard SQL.
ORDER BY UserName ASC;

# 4. Alternative (not preferred): Nested SELECT statement.
# This is equivalent to the SELECT statement above.
SELECT UserName, CNT
FROM
    # UserName and count of comments
    (SELECT UserName, COUNT(*) AS CNT
    FROM BlogComments
    GROUP BY UserName) AS UserCnt
WHERE CNT > 1
ORDER BY UserName ASC;



# 5. Count of comments for each post in descending order, and include the post id and title.
```

```
# 1. View for all BlogPosts with their BlogComments.
SELECT BlogPosts.PostId, BlogPosts.Title,
  BlogComments.Created, BlogComments.Content
FROM BlogPosts INNER JOIN BlogComments
  ON BlogPosts.PostId = BlogComments.PostId
ORDER BY BlogPosts.PostId ASC, BlogComments.Created ASC;
# 2. GROUP BY aggregation.
SELECT BlogPosts.PostId, BlogPosts.Title, COUNT(*) AS COMMENT_CNT
FROM BlogPosts INNER JOIN BlogComments
  ON BlogPosts.PostId = BlogComments.PostId
GROUP BY BlogPosts.PostId, BlogPosts.Title
# GROUP BY BlogPosts.PostId  # Non-standard SQL, relies on evaluating PK/FD for BlogPosts.Title.
ORDER BY COMMENT_CNT DESC;


# 6. UserName with most unpublished posts, including status level.
# 1. View all the unpublished posts.
SELECT BlogUsers.UserName, BlogUsers.StatusLevel,
  BlogPosts.PostId, BlogPosts.Title
FROM BlogUsers INNER JOIN BlogPosts
  ON BlogUsers.UserName = BlogPosts.UserName
WHERE Published = False
ORDER BY BlogUsers.UserName;
# 2. GROUP BY aggregation.
SELECT BlogUsers.UserName, BlogUsers.StatusLevel,
  COUNT(*) AS UNPUBLISHED_CNT
FROM BlogUsers INNER JOIN BlogPosts
  ON BlogUsers.UserName = BlogPosts.UserName
WHERE Published = False
GROUP BY BlogUsers.UserName, BlogUsers.StatusLevel
ORDER BY UNPUBLISHED_CNT DESC;
# LIMIT 1; # Or is there a tie?


# 7. The most reshared post, including post id and title.
SELECT BlogPosts.PostId, BlogPosts.Title,
  COUNT(ReshareId) AS RESHARE_CNT
FROM Reshares INNER JOIN BlogPosts
  ON Reshares.PostId = BlogPosts.PostId
GROUP BY BlogPosts.PostId, BlogPosts.Title
ORDER BY RESHARE_CNT DESC, BlogPosts.Title ASC;
# LIMIT 1;


# 8. Number of comments per day per UserName.
# You can use Date() or Cast() to convert timestamp to date:
#SELECT Date('2000-02-01 23:00:00');
#SELECT Timestamp('2000-02-01 23:00:00');
#SELECT Date(Timestamp('2000-02-01 23:00:00'));
#SELECT CAST('2000-02-01 23:00:00' AS Date);
#SELECT CAST(Timestamp('2000-02-01 23:00:00') AS Date);
SELECT UserName, Date(Created), COUNT(*) AS COUNT_PER_DAY
FROM BlogComments
GROUP BY UserName, Date(Created);  # MySQL expression in GROUP BY.
# SELECT * FROM BlogComments;
```

```
# 9. Average comments per day per UserName
SELECT UserName, AVG(COUNT_PER_DAY) AS AVG_PER_DAY
FROM (
  SELECT UserName, Date(Created), COUNT(*) AS COUNT_PER_DAY
  FROM BlogComments
  GROUP BY UserName, Date(Created)) AS PER_DAY
GROUP BY UserName;
# Nested GROUP BY references alias in SELECT clause (MySQL convenience).
SELECT UserName, AVG(COUNT_PER_DAY) AS AVG_PER_DAY
FROM (
  SELECT UserName, Date(Created) AS Created_Date, COUNT(*) AS COUNT_PER_DAY
  FROM BlogComments
  GROUP BY UserName, Created_Date) AS PER_DAY
GROUP BY UserName;


# 10. How about comment counts for all days (even for days without comments)?
# Need a "date" table for left outer join.
# 1. Identify the relevant dates.
SELECT * FROM M201502;
# 2. Identify an inner query.
SELECT *
FROM M201502 LEFT OUTER JOIN BlogComments
  ON M201502.day = Date(BlogComments.created)
ORDER BY M201502.day ASC;
# 3. Handle NULLs.
SELECT M201502.day AS DAY_VALUE,
    IF(CommentId IS NULL, 0, 1) AS DAY_HAS_COMMENT
  FROM M201502 LEFT OUTER JOIN BlogComments
    ON M201502.day = Date(BlogComments.created)
ORDER BY M201502.day ASC;
# 4. GROUP BY aggregation.
SELECT DAY_VALUE, SUM(DAY_HAS_COMMENT) AS COMMENT_SUM
FROM (
  SELECT M201502.day AS DAY_VALUE,
    IF(CommentId IS NULL, 0, 1) AS DAY_HAS_COMMENT
  FROM M201502 LEFT OUTER JOIN BlogComments
    ON M201502.day = Date(BlogComments.created)) AS COMMENT_COUNTS
GROUP BY DAY_VALUE;
# Alternatively (even better!):
SELECT M201502.day AS DAY_VALUE,
  SUM(IF(CommentId IS NULL, 0, 1)) AS COMMENT_SUM
FROM M201502 LEFT OUTER JOIN BlogComments
  ON M201502.day = Date(BlogComments.created)
GROUP BY DAY_VALUE;


# 11. UserNames with more comments than posts (hint: include all users).
# 1. Identify all the BlogUsers;
SELECT * FROM BlogUsers;
# 2. Comments per person.
SELECT UserName, COUNT(*) AS COMMENT_CNT
FROM BlogComments
```

```
GROUP BY UserName;
# 3. Posts per person.
SELECT UserName, COUNT(*) AS POST_CNT
FROM BlogPosts
GROUP BY UserName;
# 4.View of the left outer join.
SELECT BlogUsers.UserName AS B_User,
  COMMENTS.UserName AS C_USER, COMMENTS.COMMENT_CNT,
  POSTS.UserName AS P_USER, POSTS.POST_CNT
FROM BlogUsers
  LEFT OUTER JOIN (
          # Comments per user.
    SELECT UserName, COUNT(*) AS COMMENT_CNT
    FROM BlogComments
    GROUP BY UserName) AS COMMENTS
  ON BlogUsers.UserName = COMMENTS.UserName
  LEFT OUTER JOIN (
    # Posts per user.
    SELECT UserName, COUNT(*) AS POST_CNT
    FROM BlogPosts
    GROUP BY UserName) AS POSTS
  ON BlogUsers.UserName = POSTS.UserName;
# 5. Handle NULLs.
SELECT BlogUsers.UserName AS B_User,
  COMMENTS.UserName AS C_USER,
  IF(COMMENTS.UserName IS NULL, 0, COMMENTS.COMMENT_CNT) AS COMMENT_CNT_SUBTOTAL,
  POSTS.UserName AS P_USER,
  IF(POSTS.UserName IS NULL, 0, POSTS.POST_CNT) AS POST_CNT_SUBTOTAL
FROM BlogUsers
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS COMMENT_CNT
    FROM BlogComments
    GROUP BY UserName) AS COMMENTS
  ON BlogUsers.UserName = COMMENTS.UserName
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS POST_CNT
    FROM BlogPosts
    GROUP BY UserName) AS POSTS
  ON BlogUsers.UserName = POSTS.UserName;
# 6. Row filtering (selection) for outer-most query.
# A. The SELECT clause has a metric that is calculated after WHERE filtering.
#    So try another outer query.
SELECT T.UserName, T.COMMENT_CNT_SUBTOTAL, T.POST_CNT_SUBTOTAL
FROM (
          SELECT BlogUsers.UserName,
            IF(COMMENTS.UserName IS NULL, 0, COMMENTS.COMMENT_CNT) AS COMMENT_CNT_SUBTOTAL,
            IF(POSTS.UserName IS NULL, 0, POSTS.POST_CNT) AS POST_CNT_SUBTOTAL
          FROM BlogUsers
            LEFT OUTER JOIN (
                  SELECT UserName, COUNT(*) AS COMMENT_CNT
                  FROM BlogComments
                  GROUP BY UserName) AS COMMENTS
            ON BlogUsers.UserName = COMMENTS.UserName
            LEFT OUTER JOIN (
                  SELECT UserName, COUNT(*) AS POST_CNT
```

```
                FROM BlogPosts
                GROUP BY UserName) AS POSTS
        ON BlogUsers.UserName = POSTS.UserName) AS T
WHERE T.COMMENT_CNT_SUBTOTAL > POST_CNT_SUBTOTAL;
# B. Can we reduce one level of nesting?
#    The SELECT expression/alias cannot be referenced in the WHERE clause.
#    Recall WHERE filtering happens first.
#    So duplicate the SELECT expr down into WHERE filtering.
SELECT BlogUsers.UserName,
  IF(COMMENTS.UserName IS NULL, 0, COMMENTS.COMMENT_CNT) AS COMMENT_CNT_TOTAL,
  IF(POSTS.UserName IS NULL, 0, POSTS.POST_CNT) AS POST_CNT_TOTAL
FROM BlogUsers
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS COMMENT_CNT
    FROM BlogComments
    GROUP BY UserName) AS COMMENTS
  ON BlogUsers.UserName = COMMENTS.UserName
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS POST_CNT
    FROM BlogPosts
    GROUP BY UserName) AS POSTS
  ON BlogUsers.UserName = POSTS.UserName
# This would cause an unknown column error.
#WHERE COMMENT_CNT_TOTAL > POST_CNT_TOTAL;
WHERE IF(COMMENTS.UserName IS NULL, 0, COMMENTS.COMMENT_CNT)
  > IF(POSTS.UserName IS NULL, 0, POSTS.POST_CNT);
# C. Not standard SQL (and definitely not preferred):
#    MySQL allows HAVING (and GROUP BY) to reference SELECT aliases.
#    Note that WHERE cannot reference SELECT aliases.
#    Also, We know HAVING filters before SELECT is evaluated (but not in MySQL optimizations).
SELECT BlogUsers.UserName,
  IF(COMMENTS.UserName IS NULL, 0, COMMENTS.COMMENT_CNT) AS COMMENT_CNT_TOTAL,
  IF(POSTS.UserName IS NULL, 0, POSTS.POST_CNT) AS POST_CNT_TOTAL
FROM BlogUsers
  LEFT OUTER JOIN (
        SELECT UserName, COUNT(*) AS COMMENT_CNT
        FROM BlogComments
        GROUP BY UserName) AS COMMENTS
  ON BlogUsers.UserName = COMMENTS.UserName
  LEFT OUTER JOIN (
        SELECT UserName, COUNT(*) AS POST_CNT
        FROM BlogPosts
        GROUP BY UserName) AS POSTS
  ON BlogUsers.UserName = POSTS.UserName
#WHERE COMMENT_CNT_TOTAL > POST_CNT_TOTAL;  # Would cause an error.
HAVING COMMENT_CNT_TOTAL > POST_CNT_TOTAL;



#############
# EXERCISE #
#############
# 12. Compare a user's number of reshares to published posts (hint: include all users).
# Who is more likely to read than write (reshares>posts), and vice versa (reshares<posts)?
```

```
# 1. All relevant users.
SELECT * FROM BlogUsers;
# 2. Reshares per user.
SELECT UserName, COUNT(*) AS RESHARES_PER_USER
FROM Reshares
GROUP BY UserName;
# 3. Published posts per user.
SELECT UserName, COUNT(*) AS PUBLISHED_POSTS_PER_USER
FROM BlogPosts
WHERE BlogPosts.Published = True
GROUP BY UserName;
# 4. View of the left outer join.
SELECT BlogUsers.UserName AS B_User,
  RESHARES.UserName AS R_User, RESHARES.RESHARES_PER_USER,
  POSTS.UserName AS P_User, POSTS.PUBLISHED_POSTS_PER_USER
FROM BlogUsers
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS RESHARES_PER_USER
    FROM Reshares
    GROUP BY UserName) AS RESHARES
  ON BlogUsers.UserName = RESHARES.UserName
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS PUBLISHED_POSTS_PER_USER
    FROM BlogPosts
    WHERE BlogPosts.Published = True
    GROUP BY UserName) AS POSTS
  ON BlogUsers.UserName = POSTS.UserName;
# 5. Handle NULL
SELECT BlogUsers.UserName AS B_User,
  RESHARES.UserName AS R_User,
  IF(RESHARES.UserName IS NULL, 0, RESHARES.RESHARES_PER_USER) AS RESHARES_SUBTOTAL,
  POSTS.UserName AS P_User,
  IF(POSTS.UserName IS NULL, 0, POSTS.PUBLISHED_POSTS_PER_USER) AS POSTS_SUBTOTAL
FROM BlogUsers
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS RESHARES_PER_USER
    FROM Reshares
    GROUP BY UserName) AS RESHARES
  ON BlogUsers.UserName = RESHARES.UserName
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS PUBLISHED_POSTS_PER_USER
    FROM BlogPosts
    WHERE BlogPosts.Published = True
    GROUP BY UserName) AS POSTS
  ON BlogUsers.UserName = POSTS.UserName;
# 6A. Outer query for filtering.
SELECT T.UserName, T.RESHARES_SUBTOTAL, T.POSTS_SUBTOTAL
FROM (
        SELECT BlogUsers.UserName,
          IF(RESHARES.UserName IS NULL, 0, RESHARES.RESHARES_PER_USER) AS RESHARES_SUBTOTAL,
          IF(POSTS.UserName IS NULL, 0, POSTS.PUBLISHED_POSTS_PER_USER) AS POSTS_SUBTOTAL
        FROM BlogUsers
          LEFT OUTER JOIN (
                SELECT UserName, COUNT(*) AS RESHARES_PER_USER
                FROM Reshares
```

```
            GROUP BY UserName) AS RESHARES
        ON BlogUsers.UserName = RESHARES.UserName
        LEFT OUTER JOIN (
            SELECT UserName, COUNT(*) AS PUBLISHED_POSTS_PER_USER
            FROM BlogPosts
            WHERE BlogPosts.Published = True
            GROUP BY UserName) AS POSTS
        ON BlogUsers.UserName = POSTS.UserName) AS T
WHERE T.RESHARES_SUBTOTAL > T.POSTS_SUBTOTAL;
# B. Can we reduce one level of nesting?
# Duplicate SELECT expression in WHERE clause.
SELECT BlogUsers.UserName AS B_User,
  IF(RESHARES.UserName IS NULL, 0, RESHARES.RESHARES_PER_USER) AS RESHARES_SUBTOTAL,
  IF(POSTS.UserName IS NULL, 0, POSTS.PUBLISHED_POSTS_PER_USER) AS POSTS_SUBTOTAL
FROM BlogUsers
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS RESHARES_PER_USER
    FROM Reshares
    GROUP BY UserName) AS RESHARES
  ON BlogUsers.UserName = RESHARES.UserName
  LEFT OUTER JOIN (
    SELECT UserName, COUNT(*) AS PUBLISHED_POSTS_PER_USER
    FROM BlogPosts
    WHERE BlogPosts.Published = True
    GROUP BY UserName) AS POSTS
  ON BlogUsers.UserName = POSTS.UserName
WHERE IF(RESHARES.UserName IS NULL, 0, RESHARES.RESHARES_PER_USER)
  > IF(POSTS.UserName IS NULL, 0, POSTS.PUBLISHED_POSTS_PER_USER)
```