More Structured Query Language (SQL) [30 pts]

1. [10pts] List the user id, associated publisher id, and the number of degrees for all posters and order them by their number of degrees in a **descending** order. Be sure to include all posters, even those who don’t have a degree (i.e., those who have zero degrees). Expected result row(s): 8

a) [7pts] SQL Query:

```sql
SELECT P.userid, P.publisherid, COUNT(X.userid)
FROM Poster P LEFT OUTER JOIN Degree X on P.userid = X.userid
GROUP BY P.userid
ORDER BY COUNT(X.userid) DESC;
```

b) [3pts] Result:

<table>
<thead>
<tr>
<th>userid</th>
<th>publisherid</th>
<th>COUNT(X.userid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>NULL</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>NULL</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

2. [10pts] For all detectors, list their user id, account, number of articles read, and lowest “like” score that they have given an article; order the list by user id. Include all detectors, even those who have not read a single article. Expected result row(s): 11

a) [7pts] SQL Query:

```sql
SELECT U.userid, U.account, COUNT(H.userid), MIN(H.like_score)
FROM (Detector D NATURAL JOIN Users U) LEFT OUTER JOIN Has_read H ON U.userid = H.userid
GROUP BY U.userid
ORDER BY U.userid;
```
b) [3pts] Result:

<table>
<thead>
<tr>
<th>userid</th>
<th>account</th>
<th>COUNT(H.userid)</th>
<th>MIN(H.like_score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>DerrickJohn</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>ChrisHack</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>LeoHood</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>EllenPedro</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>AhmedJordan</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>SaraWallace</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>HelenCarlos</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>JedSmith</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>EllenRob</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>NihatTang</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>KellyWang</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

3. [10pts] Print a report for articles by listing each article’s user id, article id, quality, number of times the article has been read, average “like” score, and number of reports that the article has received. Include all articles in the database. Expected result row(s): 7

a) [7pts] SQL Query:

```sql
SELECT X1.userid, X1.articleid, X1.quality, X1.rds, X1.avgscore, X2.rmnds
FROM (SELECT A.userid, A.articleid, A.quality, COUNT(H.userid) as rds, AVG(H.like_score) as avgscore
FROM Article A
LEFT OUTER JOIN Has_read H
ON A.userid = H.article_userid
AND A.articleid = H.article_articleid
GROUP BY A.userid, A.articleid, A.quality) AS X1
NATURAL JOIN
(SELECT A.userid, A.articleid, A.quality, COUNT(R.userid) AS rmnds
FROM Article A
LEFT OUTER JOIN Reports R
ON A.userid = R.article_userid
AND A.articleid = R.article_articleid
GROUP BY A.userid, A.articleid, A.quality) AS X2;
```

b) [3pts] Result:

<table>
<thead>
<tr>
<th>userid</th>
<th>articleid</th>
<th>quality</th>
<th>rds</th>
<th>avgscore</th>
<th>rmnds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Clean</td>
<td>12</td>
<td>6.4167</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Clean</td>
<td>4</td>
<td>6.7500</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Clean</td>
<td>7</td>
<td>7.8571</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Clean</td>
<td>7</td>
<td>2.8571</td>
<td>5</td>
</tr>
</tbody>
</table>
Advanced SQL Features [70 pts]

4. Stored Procedures [20 pts]

a) [15 pts] Create and exercise a SQL stored procedure called NewArticle(...) that the site’s application can use to add a new article to the database. The stored procedure should automatically use the current date and time to set those fields of the new article, and it should automatically generate the new article’s article id by adding one to the number of articles that the poster currently has. (Check HW1 to see what are the default values for quality and popularity for a new article).

DELIMITER //
CREATE PROCEDURE NewArticle( /* Q4.a */
   id INTEGER,
   title VARCHAR(100),
   content VARCHAR(1000))
BEGIN
   DECLARE new_articleid INTEGER;
   SET new_articleid = (SELECT COUNT(*) FROM Article A WHERE userid = id)+1;
   INSERT INTO Article(userid,articleid,posting_datetime,popularity,quality,title,content)
   VALUES(id,new_articleid,current_timestamp(),'Regular','Clean',title,content);
END ;//
DELIMITER ;

b) [5pts] Verify that your new stored procedure is working properly by calling it as follows to add a new article and then running a SELECT query to show the procedure’s after-effects:

CALL NewArticle(2, 'WannaCry ransomware attack', 'The WannaCry ransomware attack is an ongoing cyberattack of the WannaCry ransomware cryptoworm, targeting the Microsoft Windows operating system, encrypting data and demanding ransom payments in the cryptocurrency bitcoin.');

SELECT userid, articleid, posting_datetime, popularity, quality, title
FROM Article WHERE userid = 2 /* Q4.b */;
Expected row is: 1 (Ignore null values of last row if it appears)
5. **Alter Table** [10 pts]

[10 pts] Write and execute the ALTER TABLE statement(s) needed to modify the Poster table so that whenever the Publisher associated with the poster is deleted then the Poster data is deleted too. (Note: the name of the existing foreign key constraint for publisherid is poster_ibfk_2.)

```
ALTER TABLE Poster DROP FOREIGN KEY poster_ibfk_2;
ALTER TABLE Poster ADD FOREIGN KEY (publisherid) REFERENCES Publisher(publisherid) ON DELETE CASCADE;
```

6. **Views** [20 pts]

a) [15 pts] Create the desired view (PublisherView) by writing an appropriate CREATE VIEW statement.

```
CREATE VIEW PublisherView (publisherid, name, website, number_of_articles) AS
SELECT publisherid, name, website, COUNT(articleid)
FROM (Publisher NATURAL JOIN Poster) NATURAL JOIN Article
GROUP BY publisherid, name, website;
```

b) [5 pts] Show the usefulness of your view by writing a SELECT query against the view that prints the publisher name and website for the most prolific publisher (i.e., the publisher that has published the most articles).

```
SELECT * FROM PublisherView
ORDER BY number_of_articles DESC LIMIT 1;
```

7. **Triggers** [20 pts]

a) [15 pt] Use the CREATE TRIGGER statement in MySQL to define a trigger that will do the desired job.

```
DELIMITER //
CREATE TRIGGER update_quality
AFTER INSERT ON Reports
FOR EACH ROW
BEGIN
    IF ((SELECT COUNT(*) FROM Reports
```

WHERE article_userid = NEW.article_userid AND article_articleid = NEW.article_articleid)
BETWEEN 6 AND 9)
THEN UPDATE Article SET quality = 'Suspicious'
WHERE userid = NEW.article_userid AND articleid = New.article_articleid;
ELSEIF ((SELECT COUNT(*) FROM Reports
WHERE article_userid = NEW.article_userid AND article_articleid = NEW.article_articleid) >= 10)
THEN UPDATE Article SET quality = 'Junk'
WHERE userid = NEW.article_userid AND articleid = New.article_articleid;
END IF;
END;://
DELIMITER ;
b) [5 pts] Execute the following INSERT and then SELECT statements to show the effect of your trigger:
INSERT INTO Reports VALUES(21,3,2,'Kinda incorrect'), (10,4,1,'Team is not correct');
SELECT userid, articleid, quality FROM Article /* Q7.b */;
userid | articleid | quality
--- | --- | ---
1 | 1 | Clean
1 | 2 | Clean
2 | 1 | Clean
3 | 1 | Clean
3 | 2 | Suspicious
4 | 1 | Junk
6 | 1 | Junk
9 | 1 | Clean

8. **Extra Credit** [10 pts]

a) [5 pts] Create and show this additional trigger in the same manner as in problem 7.
DELIMITER //
CREATE TRIGGER update_quality2
AFTER DELETE ON Reports
FOR EACH ROW
BEGIN
  IF ((SELECT COUNT(*) FROM Reports
       WHERE article_userid = OLD.article_userid AND article_articleid = OLD.article_articleid)
      BETWEEN 6 AND 9)
  THEN UPDATE Article SET quality = 'Suspicious'
      WHERE userid = OLD.article_userid AND articleid = OLD.article_articleid;
  ELSEIF ((SELECT COUNT(*) FROM Reports
            WHERE article_userid = OLD.article_userid AND article_articleid = OLD.article_articleid) < 6)
  THEN UPDATE Article SET quality = 'Clean'
      WHERE userid = OLD.article_userid AND articleid = OLD.article_articleid;
  END IF;
END;://
DELIMITER ;

b) [5 pts] Create and execute DELETE statements for the inserted tuples in question 7.b to verify that your trigger indeed works.

Delete entries (21,3,2) and (10,4,1). (Write delete statements for these entries.)

SELECT userid, articleid, quality FROM Article /* Q8.b */;
DELETE FROM Reports WHERE userid = 21 AND article_userid = 3 AND article_articleid = 2;
DELETE FROM Reports WHERE userid = 10 AND article_userid = 4 AND article_articleid = 1;

<table>
<thead>
<tr>
<th>userid</th>
<th>articleid</th>
<th>quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Clean</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Clean</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Suspicious</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>Junk</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>Clean</td>
</tr>
</tbody>
</table>