Solution to queries and updates in SQL assignment

The boat reservations database has the following schema:

- **sailor**: `sname` (string), `rating` (integer)
- **boat**: `bname` (string), `color` (string), `rating` (integer)
- **reservation**: `sname` (string), `bname` (string), `day` (string), `start` (integer), `finish` (integer)

The rating attribute for boats indicates the minimum rating required of a sailor reserving the boat.

(c) Write the following queries in SQL (of course, the queries must work on all data, not just the sample one):

1. List all pairs of sailors and boats they are qualified to sail.

   ```sql
   select sname, bname
   from sailor, boat
   where sailor.rating ≥ boat.rating
   ```

2. List, for each sailor, the number of boats they are qualified to sail\(^1\).

   ```sql
   (select sname, count(bname) as number
   from sailor, boat
   where sailor.rating ≥ boat.rating
   group by sname)
   union
   (select sname, 0 as number
   from sailor
   where sname not in
   (select sname
   from sailor, boat
   where sailor.rating ≥ boat.rating))
   ```

\(^1\)Note that Brutus and Bob are not qualified to sail any boat, so their count should be shown as zero.
3. List the sailors with the lowest rating. Provide two queries:

(i) one using the MIN aggregate function, and

\[
\text{select sname}
\text{from sailor}
\text{where rating in}
\quad (\text{select MIN(rating) from sailor})
\]

(ii) another without using MIN.

\[
\text{select s.sname}
\text{from sailor s}
\text{where not exists}
\quad (\text{select *}
\text{from sailor}
\text{where rating < s.rating})
\]

4. List the sailors who have at least one reservation and only reserved red boats.

\[
\text{select distinct sname}
\text{from reservation}
\text{where sname not in}
\quad (\text{select sname}
\text{from reservation, boat}
\text{where reservation.bname = boat.bname and boat.color \neq 'red'})
\]

This uses the fact that bname is a key in boat. Otherwise we would write:

\[
\text{select distinct sname}
\text{from reservation}
\text{where sname not in}
\quad (\text{select sname from reservation where bname not in}
\quad (\text{select bname from boat where color = 'red'})
\]


5. List the sailors who reserved no red boat\(^2\).

```sql
select s.sname
from sailor s
where not exists
  (select * from reservation, boat
   where sname = s.sname and reservation.bname = boat.bname
   and color = 'red')
```

6. List the sailors who reserved all red boats. Provide three SQL queries, using nested sub-queries in different ways:

- with NOT IN tests;

```sql
select sname
from sailor
where sname not in
  (select sname from sailor, boat b
   where b.color = 'red' and sname not in
    (select sname from reservation
     where bname = b.bname ))
```

- with NOT EXISTS tests;

```sql
select s.sname
from sailor s
where not exists
  (select * from boat b
   where b.color = 'red' and not exists
    (select * from reservation
     where sname = s.sname and reservation.bname = b.bname ))
```

\(^2\)Note that Brutus is one of these since he has no reservation at all.
– with COUNT aggregate functions.

```sql
select s.sname
from sailor s
where
(select count(*) from boat b
 where b.color = 'red' and
 (select count(*) from reservation
  where sname = s.sname and bname = b.bname) = 0 ) = 0
```

Note that, if there are no red boats, then all sailors are in the answer (which is the correct semantics).

7. For each reserved boat, list the average rating of sailors having reserved that boat. **Caution:** watch for dangerous duplicate tuples!

```sql
create view unique as
select distinct sailor.sname, rating, bname
from sailor, reservation
where sailor.sname = reservation.sname;
select bname, avgrating = avg(rating * 1.0)
from unique
group by bname
```

The reason for multiplying rating with 1.0 in `avg(rating * 1.0)` is to convert the rating to a real so that the average is not truncated to an integer. The role of the view `unique` is to provide the list of boats and sailors with their ratings who have reserved the boat without duplicates. This is to avoid counting the same sailor multiple times if the sailor has multiple reservations of the same boat. For example, the following alternative does not avoid this problem:

```sql
select bname, avgrating = avg(rating)
from sailor, reservation
where sailor.sname = reservation.sname
group by bname
```
(d) Formulate and run a query to verify that there are no conflicting reservations, i.e. there are no reservations for the same boat in the same day for strictly overlapping time intervals. Specifically, the query should look for violations. If there are violations, it should return the pairs of reservations that violate the constraints. The answer should be a table with schema

<table>
<thead>
<tr>
<th>bname</th>
<th>day</th>
<th>sname1</th>
<th>start1</th>
<th>finish1</th>
<th>sname2</th>
<th>start2</th>
<th>finish2</th>
</tr>
</thead>
</table>

In the answer, *bname* is a boat for which there are conflicting reservations, *day* is the day of two conflicting reservations, *sname1*, *start1*, *finish1*, are the sailor name, start time and end time for the first reservation and *sname2*, *start2*, *finish2* similarly for the second reservation. The answer should contain just one tuple for each conflict (e.g., for the example data, only one of the tuples

\[ \langle \text{Interlake, Monday, Andy, 10, 14, Rusty, 9, 11} \rangle \]

and

\[ \langle \text{Interlake, Monday, Rusty, 9, 11, Andy, 10, 14} \rangle \]

should be included in the answer). The answer should be empty if there is no violation.

```
select x.bname, x.day,
    x.sname as sname1, x.start as start1, x.finish as finish1,
    y.sname as sname2, y.start as start2, y.finish as finish2
from reservation x, reservation y
where x.bname = y.bname and x.day = y.day and
    (x.start = y.start and x.finish = y.finish and x.sname < y.sname) or
    (x.start = y.start and x.finish > y.finish) or
    (x.start < y.start and y.start < x.finish))
```

Note that this lists only one tuple for every pair of conflicting reservations *x*, *y*. Indeed, suppose first the time intervals of the reservations are the same. Then the sailors of the two reservations should be different (otherwise *x* and *y* would be the same reservation). The double listing is avoided by requiring that *x*.sname < *y*.sname (this compares the strings). If the starting time of the two reservations are the same
and the intervals are not equal then only one of them ends first, so requiring $x.\text{finish} > y.\text{finish}$ avoids again the double listing. Finally, if the reservations have different starting times then only one starts first, so $x.\text{start} < y.\text{start}$ avoids double listing.

(e) Formulate and execute the following updates in SQL.

1. Change all red boats to blue and all blue boats to red, without explicitly naming the boats involved.

   ```sql
   update boat
   set color = 'xxx' where color = 'blue';
   update boat
   set color = 'blue' where color = 'red';
   update boat
   set color = 'red' where color = 'xxx'
   ``

2. Delete all sailors who are not qualified to sail any boat.

   ```sql
   delete from reservation r
   where not exists
   (select * from boat, sailor
   where sname = r.sname and sailor.rating >= boat.rating);
   delete from sailor s
   where not exists
   (select * from boat
   where s.rating >= boat.rating)
   ```

Note: the first deletion from reservation prevents violation of the referential integrity constraint from reservations to sailor when a sailor is deleted.