(A) Write the following queries in SQL

1. List all boats reserved on Wednesday and their color.

   ```
   select distinct reservation.bname, color
   from reservation, boat
   where reservation.bname = boat.bname and reservation.day = 'Wednesday'
   ```

2. List all pairs of sailors who have reserved boats on the same day (avoid listings of the form \( \langle a, a \rangle \)), or listing both \( \langle a, b \rangle \) and \( \langle b, a \rangle \)).

   ```
   select distinct x.sname, y.sname
   from reservation x, reservation y
   where x.day = y.day and x.sname < y.sname
   ```

3. For each day, list the number of red boats reserved on that day. If there is no red boat reserved on a given day, the number shown should be zero. In particular, if a day of the week does not appear at all in the `reservation` relation, it should appear in the answer with zero red boat reservations.

   ```
   select r.day, count(distinct r.bname) as number
   from reservation r, boat b
   where r.bname = b.bname and b.color = 'red'
   group by day
   union
   select day, 0 as number
   from weekday
   where day not in
   (select day
    from reservation, boat
    where reservation.bname = boat.bname
    and color = 'red')
   ```
4. List the days appearing in the reservation table for which only red boats are reserved. Provide three SQL queries, using nested sub-queries in different ways:

   – with only NOT IN tests;

     ```sql
     select distinct day
     from reservation
     where day not in
     (select r.day
      from reservation r, boat b
      where r.bname = b.bname and b.color <> 'red')
     ```

   – with only NOT EXISTS tests;

     ```sql
     select distinct x.day
     from reservation x
     where not exists
     (select * from reservation r, boat b
      where r.day = x.day and r.bname = b.bname and b.color <> 'red')
     ```

   – with only COUNT aggregate functions.

     ```sql
     select distinct x.day
     from reservation x
     where
     (select count(*) from reservation r, boat b
      where r.day = x.day and r.bname = b.bname and b.color <> 'red') = 0
     ```

5. For each day of the week occurring in the reservation relation, list the average rating of sailors having reserved boats that day. **Caution:** watch for dangerous duplicate tuples!. You may write this as a sequence of two or more queries (including CREATE VIEW if useful). Make sure the average is not truncated to an integer (you may convert an integer to a real by multiplying the integer with 1.0).
create view UniqueRatings as
select distinct sailor.sname, rating, day
from sailor, reservation
where sailor.sname = reservation.sname;
select day, avgrating as avg(rating * 1.0)
from UniqueRatings
group by day

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 UniqueRatings is to provide the list of sailors and their ratings for each day without duplicates. This is to avoid counting the same sailor multiple times if the sailor has multiple reservations in the same day. For example, the following alternative does not avoid this problem:

select day, avgrating as avg(rating)
from sailor, reservation
where sailor.sname = reservation.sname
group by day

6. List the busiest days, i.e. the days with the largest number of reservations. Note: if the reservations table is empty, all days of the week should be in the answer.

Here is one possibility. Note that all days are in the answer if reservation is empty.

create view NumPerDay as
select day, count(*) as num
from reservation
group by day;
select day from NumPerDay
where num in
    (select max(num) from NumPerDay)
union
select day from weekday
where not exists (select * from reservation)
7. Formulate a query to verify that all sailors having reservations are qualified to sail the boats they reserved. Specifically, the query should look for violations. It should return the sailor/boat pairs that violate the constraint together with the rating of the sailor (named \textit{srating}), the rating of the boat (named \textit{brating}), and the day of the reservation. The answer should be empty if there is no violation.

\begin{verbatim}
select r.day, s.sname, b.bname, s.rating as srating, b.rating as brating 
from sailor s, reservation r, boat b 
where s.sname = r.sname and r.bname = b.bname and s.rating < b.rating
\end{verbatim}

(B) Write the following updates in SQL.

1. Switch all Wednesday and Monday reservations, without explicitly naming the boats involved. \textbf{Hint}: You may use several commands to do this.

\begin{verbatim}
update reservation
set day = 'xxx' where day = 'Monday';

update reservation
set day = 'Monday' where day = 'Wednesday';

update reservation
set day = 'Wednesday' where day = 'xxx'
\end{verbatim}

2. Delete all sailors who are not qualified to sail any boat, together with their reservations (this has to be done carefully, to avoid violations of the foreign key constraint from \textit{reservation} to \textit{sailor}).
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 foreign key constraint from reservations to sailor when a sailor is deleted.