Problem 1. True or false (no justification required):

(i) If A is an attribute of type integer in a table, then \(A \times 0\) always evaluates to 0 in SQL.

- \text{Circle one:} \quad \text{False (when A is null)}

(ii) Aliases (tuple variables) in SQL do not increase the power of the language and are provided just for convenience.

- \text{Circle one:} \quad \text{False}

(iii) In SQL, all nested queries without correlated subqueries can be unnested.

- \text{Circle one:} \quad \text{False}

(iv) The natural join in SQL does not add power to the language and is provided just for convenience.

- \text{Circle one:} \quad \text{True}

(v) SQL can express some queries that relational calculus cannot express.

- \text{Circle one:} \quad \text{True (e.g. group-by and aggregates)}

(vi) Universal quantification is redundant in relational calculus (it can be simulated by the other operators).

- \text{Circle one:} \quad \text{True}

(vii) In SQL, the \texttt{exists} clause can be simulated with the \texttt{count} function.

- \text{Circle one:} \quad \text{True}

(viii) Employee is a relation with attributes \textit{name}, \textit{salary}. The SQL query

\begin{verbatim}
select name from employee
where salary = salary
\end{verbatim}

always produces the same answer as the query
select name from employee

– Circle one: False (if salary = null then salary = salary evaluates to unknown)

(ix) Employee is a relation with attributes name, salary. The SQL query

select name from employee
where salary = salary OR 0 = 0

always produces the same answer as the query

select name from employee

– Circle one: True (0 = 0 is always true and unknown or true evaluates to true)

(x) \( R \) is a relation with a single attribute A. The query

select A from R
group by A

always produces the same answer as the query

select A from R

– Circle one: False (when \( R \) contains duplicates)
**Problem 2 (17 points)**
The simplified boat reservations database has the following schema:

- **sailor**: `sname` (string), `rating` (integer)
- **boat**: `bname` (string), `rating` (integer)
- **reservation**: `sname` (string), `bname` (string)

The rating attribute for boats indicate the minimum rating required of a sailor in order to sail the boat. In addition, the following hold:

- `sname` is the primary key of `sailor` (so every sailor has just one rating);
- `bname` is the primary key of `boat` (so every boat has just one rating);
- `bname` is a non-null foreign key in the `reservation` relation, referencing relation `boat` (so every `bname` in `reservation` occurs in `boat`, but the converse need not be true)
- `sname` is a non-null foreign key in the `reservation` relation, referencing relation `sailor` (so every `sname` in `reservation` occurs in `sailor`, but the converse need not be true)

Here is one example instance over the above schema:

<table>
<thead>
<tr>
<th>sailor</th>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Horatio</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Rusty</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boat</th>
<th>bname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bay</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Interlake</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Marine</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>SpeedQueen</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reservation</th>
<th>sname</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>Interlake</td>
<td></td>
</tr>
<tr>
<td>Andy</td>
<td>Bay</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>SpeedQueen</td>
<td></td>
</tr>
<tr>
<td>Horatio</td>
<td>Marine</td>
<td></td>
</tr>
<tr>
<td>Rusty</td>
<td>Bay</td>
<td></td>
</tr>
<tr>
<td>Rusty</td>
<td>Interlake</td>
<td></td>
</tr>
</tbody>
</table>
For each of the following queries, express the query as indicated.

(a) (5 points) *Find the boats reserved by at least one sailor qualified to sail the boat*

For example, the answer on the sample data is:

<table>
<thead>
<tr>
<th>Answer</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bay</td>
<td></td>
</tr>
<tr>
<td>Interlake</td>
<td></td>
</tr>
<tr>
<td>Marine</td>
<td></td>
</tr>
</tbody>
</table>

Note that *SpeedQueen* is not in the answer because Bob is not qualified to sail the boat.

Write the query in (i) tuple calculus and (ii) SQL.

**Answers**

(i) tuple calculus

\[
\begin{align*}
\{ x : bname &| \exists b \in boat \ \exists r \in reservation \ \exists s \in sailor \\
( x(bname) = b(bname) \land r(bname) = b(bname) \land r(sname) = s(sname) \land s(rating) \geq b(rating) ) \} 
\end{align*}
\]

(ii) SQL

```
select b.bname
from boat b, reservation r, sailor s
where r.bname = b.bname and r.sname = s.sname and s.rating >= b.rating
```
NAME:____________________

(b) (12 points) Find the boats reserved by every sailor qualified to sail the boat

For example, the answer on the sample data is:

<table>
<thead>
<tr>
<th>Answer</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interlake</td>
<td></td>
</tr>
<tr>
<td>SpeedQueen</td>
<td></td>
</tr>
</tbody>
</table>

*Interlake* is in the answer because it is reserved by Andy and Rusty, and these are all sailors qualified to sail it. *SpeedQueen* is in the answer because no sailor is qualified to sail it (so it is reserved by every sailor qualified to sail it). *Bay* is not in the answer because Horatio did not reserve it (although he is qualified to sail it). Similarly, *Marine* is not in the answer because Andy and Rusty did not reserve it (although both are qualified to sail it).

For this query, do the following:

(i) (1 point) determine whether the query is monotonic (explain)

(ii) (4 points) express the query in tuple calculus, using \( \exists \) and \( \forall \);

(iii) (3 points) rewrite the query in (i) using only \( \exists \)

(iv) (4 points) write the SQL query corresponding directly to the tuple calculus query in (ii), using **not exists** tests on nested queries.

**Answers**

(i) is the query monotonic (explain)?

The query is **not** monotonic. Consider the instance above and the answer to the query. Suppose we add to the sailor relation a sailor Jane with rating 10. Then Jane is qualified to sail all boats, but since no boat is reserved by her, the answer to the query will be empty.

(ii) tuple calculus using \( \exists \) and \( \forall \)

\[
\{ x : bname \mid \exists b \in boat(x(bname) = b(bname)) \land \forall s \in sailor(s(rating) \geq b(rating) \rightarrow \exists r \in reservation(r(bname) = b(bname) \land r(sname) = s(sname))) \}
\]
(iii) query in (ii) rewritten to use only $\exists$

$$\{ x : bname \mid \exists b \in boat(x(bname) = b(bname)) \land \neg \exists s \in sailor(s(rating) \geq b(rating)) \land \neg \exists r \in reservation(r(bname) = b(bname) \land r(sname) = s(sname)))) \}$$

(iv) SQL query corresponding directly to the tuple calculus query in (iii), using not exists tests on nested queries.

```sql
select b.bname
from boat b
where not exists
  (select *
    from sailor s
    where s.rating >= b.rating and not exists
      (select * from reservation r
        where r.bname = b.bname and r.sname = s.sname ))
```
Problem 3 (10 points) Consider a university registrar database containing a relation

Record: ID (string), Year (integer), Qtr (string), Course (string)

The relation records, for each student ID, calendar year and quarter (F, W or S), the courses taken by the student. We define academic year \( N \) to consist of 3 quarters: F of calendar year \( N \), W of calendar year \( N + 1 \), and S of calendar year \( N + 1 \) (for example, the academic year 2014 consists of F 2014, W 2015, and S 2015). Assume for simplicity that a student is enrolled in consecutive academic years and takes at least one course per academic year. However, a student need not take courses every quarter. For example, the record of student A123 might consist of the following tuples:

<table>
<thead>
<tr>
<th>Record</th>
<th>ID</th>
<th>Year</th>
<th>Qtr</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>A123</td>
<td>2013</td>
<td>W</td>
<td>CSE8</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2013</td>
<td>S</td>
<td>CSE10</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2013</td>
<td>F</td>
<td>CSE101</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2015</td>
<td>W</td>
<td>CSE120</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2015</td>
<td>W</td>
<td>CSE121</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2015</td>
<td>F</td>
<td>CSE105</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2016</td>
<td>W</td>
<td>CSE132A</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2016</td>
<td>W</td>
<td>CSE132B</td>
<td></td>
</tr>
</tbody>
</table>

Create a view CourseLoad with attributes ID, Acad-Year, Num-courses, that provides for each student ID the number of courses taken each academic year the student was enrolled. For example, the view should produce, for student A123 with the above record, the following tuples:

<table>
<thead>
<tr>
<th>CourseLoad</th>
<th>ID</th>
<th>Acad-Year</th>
<th>Num-courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>A123</td>
<td>2012</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2013</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2014</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>A123</td>
<td>2015</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

You may define intermediate views if useful.
Answer to Problem 3

Here is one possibility. Create first a view *Acad-Year-Record* providing the courses taken by each student in each academic year:

```
create view Acad-Year-Record as
(select ID, Year as Acad-Year, Course
from Record where Qtr = 'F')
union
(select ID, Year - 1 as Acad-Year, Course
from Record where Qtr = 'W' or Qtr = 'S')
```

Now define the desired view as follows:

```
create view CourseLoad as
select ID, Acad-Year, count(Course)
from Acad-Year-Record
group by ID, Acad-Year
```

Problem 4. (4 points) Consider the relation (from the second Gradiance Lab):

     Ships(name, yearLaunched, country, numGuns, gunSize, displacement)

Write an SQL query that lists the battleships whose guns have the second largest gun size. More precisely, find the ships whose gun size was exceeded by only one gun size, no matter how many other ships had that larger gun size. List the names of the ships and their gun size.

**Solution** (provided by Gradiance):

```
SELECT name, gunSize FROM Ships WHERE gunSize =
(SELECT MAX(gunSize)
FROM SHIPS WHERE gunSize <> (SELECT MAX(gunSize) FROM Ships))
```
Problem 5. (3 points) (Gradiance practice homework) The table below gives the scores in the Japanese Baseball League for two consecutive days. The Opponent is NULL if the Team did not play on that day. The number of Runs is given as NULL if either the team did not play, or will play on that day but the game is not yet concluded.

<table>
<thead>
<tr>
<th></th>
<th>Team</th>
<th>Day</th>
<th>Opponent</th>
<th>Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dragons</td>
<td>Sunday</td>
<td>Swallows</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Tigers</td>
<td>Sunday</td>
<td>BayStars</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Carp</td>
<td>Sunday</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>Swallows</td>
<td>Sunday</td>
<td>Dragons</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>BayStars</td>
<td>Sunday</td>
<td>Tigers</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Giants</td>
<td>Sunday</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>Dragons</td>
<td>Monday</td>
<td>Carp</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>Tigers</td>
<td>Monday</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>Carp</td>
<td>Monday</td>
<td>Dragons</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>Swallows</td>
<td>Monday</td>
<td>Giants</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>BayStars</td>
<td>Monday</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>Giants</td>
<td>Monday</td>
<td>Swallows</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

What is the result of executing on this data the query:

SELECT Team, Day 
FROM Scores  
WHERE Opponent IS NULL OR 
    NOT (Runs ≥ 0)

**Solution**

The first condition in the WHERE clause asks for all rows where the Opponent column is NULL (rows 3, 6, 8, and 11) or the Runs column is at least 0. Even though we know that it is impossible to score a negative number of runs, the truth value of "Runs ≥ NULL" is "unknown". Thus, the truth value of "NOT Runs ≥ NULL" is also "unknown", and the second condition of the WHERE clause is not “true” for any of the rows. Thus, only the four rows with NULL in the Opponent column meet the condition. The resulting output is:

<table>
<thead>
<tr>
<th>Team</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carp</td>
<td>Sunday</td>
</tr>
<tr>
<td>Giants</td>
<td>Sunday</td>
</tr>
<tr>
<td>Tigers</td>
<td>Monday</td>
</tr>
<tr>
<td>Bay Stars</td>
<td>Monday</td>
</tr>
</tbody>
</table>
Problem 6. (6 points) In a tree $T$, two distinct nodes $a$ and $b$ are said to be “of the same generation” if they are at the same distance from their closest common ancestor. Consider a query whose input is a relation $T$ with attributes $A,B$ (of type char), in which tuples represent the edges of the tree, and whose output is a relation $Same-generation$ with attributes $E,F$ such that $(e,f)$ belongs to the answer iff $e$ and $f$ are nodes of $T$ of the same generation. For example, on input

$$
\begin{array}{|c|c|}
\hline
T & A & B \\
\hline
r & a \\
r & f \\
a & b \\
a & c \\
b & d \\
c & e \\
f & g \\
f & h \\
g & i \\
\hline
\end{array}
$$

representing the tree

```
       r
       / \  
      a   f
     / \  / \
   b   c   g   h
    / \    / \
   d   e  f   i
```

the answer to the query is

$$
\begin{array}{|c|c|}
\hline
Same-generation & E & F \\
\hline
a & f \\
b & c \\
g & h \\
b & g \\
b & h \\
c & g \\
c & h \\
d & e \\
d & i \\
e & i \\
\hline
\end{array}
$$

(repetitions of the same pair, such as $(a,f)$ and $(f,a)$, are omitted). Write an SQL create recursive view statement that computes the $Same-generation$ query (without repetitions of the same pair).

create recursive view SG as
( select t1.B as E, t2.B as F
from T t1, T t2
where t1.A = t2.A and t1.B < t2.B)
union
( select t1.B as E, t2.B as F
from SG s, T t1, T t2
where t1.A = s.E and t2.A = s.F)