Problem 1 (5 points) True or false (no justification required):

(i) If A is an attribute of type integer in a table, then A*0 always evaluates to 0 in SQL.
   - Circle one: False (think of nulls)

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

(iii) In SQL, all nested queries without correlated subqueries can be unnested.
   - Circle one: False

(iv) Nested queries in the from clause of SQL queries are provided just for convenience and can always be eliminated.
   - Circle one: True

(v) SQL can express some queries that relational calculus cannot express.
   - Circle one: True

(vi) Universal quantification is redundant in relational calculus (it can be simulated by the other operators).
   - Circle one: True

(vii) In SQL, the exists clause can be simulated with the count function.
   - Circle one: True

(viii) Employee is a relation with attributes name, salary. The SQL query
select name from employee
where salary = salary

always produces the same answer as the query

select name from employee

- Circle one: False (think of null salaries)

(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 (because 0 = 0 evaluates to true, and (anything 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 (the group by eliminates duplicates)
Problem 2 (3 points) (Gradiance practice homework) Consider the relations R, S and T below:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th></th>
<th>C</th>
<th>D</th>
<th></th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Compute the result of the query:

```sql
SELECT A, F, SUM(C), SUM(D)
FROM R, S, T
WHERE B = C AND D = E
GROUP BY A, F
HAVING COUNT(*) > 1
```

Solution

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>SUM(C)</th>
<th>SUM(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Problem 3 (3 points) Consider the movie database with relations

| schedule | theater | title | movie | title | director | actor |

Write an SQL query without joins whose result is

\[
\text{schedule left outer join movie}
\]

This essentially shows that join is syntactic sugar in SQL.

Solution

```sql
select * from schedule, movie
where schedule.title = movie.title
union
select theater, title, null as director, null as actor
from schedule
where title not in (select title from movie)
```
Problem 4 (14 points) The simplified boat reservations database has the following schema:

\begin{align*}
\textit{sailor}: & \text{ sname (string), rating (integer)} \\
\textit{boat}: & \text{ bname (string), rating (integer)} \\
\textit{reservation}: & \text{ sname (string), bname (string), day (string)} \\
\textit{weekday}: & \text{ day (string)}
\end{align*}

Relation \textit{weekday} contains the days of the week. The rating attribute for boats indicate the minimum rating required of a sailor in order to sail the boat. In addition, the following hold:

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

Here is one example instance over the above schema:

\begin{center}
\begin{tabular}{|l|l|l|}
\hline
\textit{sailor} & \textit{sname} & \textit{rating} \\
\hline
Andy & 8 & \\
Bob & 1 & \\
Brutus & 1 & \\
Horatio & 7 & \\
Rusty & 8 & \\
\hline
\end{tabular}
\hspace{1cm}
\begin{tabular}{|l|l|l|}
\hline
\textit{boat} & \textit{bname} & \textit{rating} \\
\hline
Bay & 3 & \\
Interlake & 8 & \\
Marine & 7 & \\
SpeedQueen & 9 & \\
\hline
\end{tabular}
\hspace{1cm}
\begin{tabular}{|l|l|l|l|}
\hline
\textit{reservation} & \textit{sname} & \textit{bname} & \textit{day} \\
\hline
Andy & Interlake & Tuesday & \\
Andy & Bay & Monday & \\
Bob & SpeedQueen & Tuesday & \\
Horatio & Marine & Sunday & \\
Rusty & Bay & Monday & \\
Rusty & Interlake & Wednesday & \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textit{weekday} & \textit{day} \\
\hline
Monday & \\
Tuesday & \\
Wednesday & \\
Thursday & \\
Friday & \\
Saturday & \\
Sunday & \\
\hline
\end{tabular}
\end{center}
We say that a reservation is valid if the sailor is qualified to sail the reserved boat.

For each of the following queries, express the query as indicated.

(a) (5 points) Find the sailors who have at least one valid reservation

For example, the answer on the sample data is:

<table>
<thead>
<tr>
<th>Answer</th>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Andy</td>
</tr>
<tr>
<td></td>
<td>Horatio</td>
</tr>
<tr>
<td></td>
<td>Rusty</td>
</tr>
</tbody>
</table>

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

For this query, do the following:

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

   The query is monotonic. Once a sailor has a valid reservation, this will continue to be true if data is added.

(ii) (2 points) write the query in tuple calculus

   \[
   \{ n : sname \mid \exists s \in \text{sailor} \ \exists r \in \text{reservation} \ \exists b \in \text{boat} \ (n(sname) = s(sname) \land r(sname) = s(sname) \land r(bname) = b(bname) \land s(rating) \geq n(rating)) \}
   \]

(iii) (2 points) write the query in SQL

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

(b) (9 points) Find the days of the week for which all reservations made on that day are valid

For example, the answer on the sample data is:

<table>
<thead>
<tr>
<th>Answer</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Monday</td>
</tr>
<tr>
<td></td>
<td>Wednesday</td>
</tr>
<tr>
<td></td>
<td>Thursday</td>
</tr>
<tr>
<td></td>
<td>Friday</td>
</tr>
<tr>
<td></td>
<td>Saturday</td>
</tr>
<tr>
<td></td>
<td>Sunday</td>
</tr>
</tbody>
</table>

Tuesday is not in the answer because it has an invalid reservation (by Bob for SpeedQueen). Thursday, Friday, Saturday are in the answer because there are no reservations on those days, so all reservations are valid.

For this query, do the following:

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

The query is not monotonic. In the example above, if an invalid reservation is added on Thursday, this day is no longer in the answer.

(ii) (4 points) express the query in tuple calculus, using ∃ and ∀;

\[ \{ d : day \mid \text{weekday}(d) \land \forall r \in \text{reservation} (r(day) = d(day) \rightarrow \exists s \in \text{sailor} \exists b \in \text{boat} (s(sname) = r(sname) \land b(bname) = r(bname) \land s(rating) \geq b(rating))) \} \]

(iii) (1 points) rewrite the query in (ii) using only ∃

\[ \{ d : day \mid \text{weekday}(d) \land \neg \exists r \in \text{reservation} (r(day) = d(day) \land \neg \exists s \in \text{sailor} \exists b \in \text{boat} (s(sname) = r(sname) \land b(bname) = r(bname) \land s(rating) \geq b(rating))) \} \]
(iv) (3 points) write the SQL query corresponding directly to the tuple calculus query in (iii), using only **not exists** tests on nested queries.

```sql
select d.day from weekday d
where not exists
  (select * from reservation r
   where r.day = d.day and not exists
     (select * from sailor s, boat b
      where s.sname = r.sname and b.bname = r.bname and s.rating >= b.rating ))
```
NAME:_____________________

Problem 5 (Gradiance inspired) (3 points) For the database in Problem 4, write an SQL query that finds the second busiest days of the week (i.e. the days that have the second highest number of reservations). For example, the answer on the sample database is Sunday, Wednesday, which have one reservation each, second after Monday and Tuesday, which have two reservations each. If Sunday and Wednesday also had two reservations each, then the second busiest days would be Thursday, Friday, Saturday, which have zero reservations each.

Here is a possibility.

1. First compute the number of reservations for each day of the week:
   
   create view Numres(day, num) as
   select day, count(*) as num
   from reservation
   group by day
   union
   select day, 0 as num
   from weekday where day not in (select day from reservation)

2. Now find the second busiest days:
   
   select day from Numres where num =
   (select max(num) from Numres where num < (select max(num) from Numres))
Problem 6 (2 points) For what values of \(x\) and \(y\), including NULL, does the Boolean expression

\[ x \leq 3 \text{ AND NOT}(y > 1) \]

have the truth value UNKNOWN in SQL?

Solution
The AND evaluates to UNKNOWN when at least one of conjuncts is UNKNWON and none is FALSE. This corresponds to the following values for \(x\) and \(y\):

- \(x\) is NULL and \(y\) is NULL
- \(x\) is NULL and \(y < 1\) (so NOT \((y > 1)\) is true)
- \(x \leq 3\) and \(y\) is NULL
Problem 7 (5 points) Consider a query whose input is a relation $T$ with attributes $A, B$ (of type char), in which tuples represent the edges of a directed tree, and whose output is a relation Least-Common-Ancestor with attributes $E, F, G$ such that $(e, f, g)$ belongs to the answer iff $e$ and $f$ are distinct nodes of the tree, $e$ and $f$ are not descendants or ancestors of each other, and $g$ is their closest common ancestor. For example, on input

<table>
<thead>
<tr>
<th>$T$</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>$a$</td>
<td></td>
</tr>
<tr>
<td>$r$</td>
<td>$f$</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$c$</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$d$</td>
<td></td>
</tr>
</tbody>
</table>

representing the tree

\[
\begin{array}{cl}
    \text{r} & \text{a} \\
    \text{a} & \text{f} \\
    \text{a} & \text{c} \\
    \text{b} & \text{d} \\
\end{array}
\]

the answer to the query is

<table>
<thead>
<tr>
<th>Least-Common-Ancestor</th>
<th>$E$</th>
<th>$F$</th>
<th>$G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$f$</td>
<td>$r$</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$f$</td>
<td>$r$</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td></td>
</tr>
<tr>
<td>$d$</td>
<td>$c$</td>
<td>$a$</td>
<td></td>
</tr>
<tr>
<td>$d$</td>
<td>$f$</td>
<td>$r$</td>
<td></td>
</tr>
<tr>
<td>$c$</td>
<td>$f$</td>
<td>$r$</td>
<td></td>
</tr>
</tbody>
</table>

(this omits repetitions such as $(a, f, r)$ and $(f, a, r)$). Write an SQL query using a with recursive clause that computes the Least-Common-Ancestor query (without repetitions $(x, y, z)$ and $(y, x, z)$). Write your answer on the next page. **Hint:** the with recursive clause should define a temporary table providing descendants of each node, which can then be used to find the least common ancestors.
Solution

In this problem the recursion is very easy (the descendant relation D is just the transitive closure of T) but the way D is used is slightly trickier.

with recursive D(A,B) as
  select * from T
  union
  select T.A, D.B from T, D
  where T.B = D.A
% end of the recursive definition of D

select x.B as E, y.B as F, x.A as G
from T x, T y
where x.A = y.A and x.B < y.B
union
select u.B as E, v.B as F, x.A as G
from T x, T y, D u, D v

The first part of the union says that the LCA of two children x.B and y.B of the same node x.A = y.A is that node. The second part says that if in addition u.B is a descendant of x.B and v.B is a descendant of y.B, then their LCA is also the parent of x.B and y.B (that is, x.A = y.A).