Some Practice Problems (and some solutions)

Exercise 1 (SQL and Relational Algebra)
Consider the following database schema:

\[
\begin{align*}
\text{Frequents}(\text{Drinker}, \text{Bar}) & \quad \text{Serves}(\text{Bar}, \text{Beer}) \\
\text{Likes}(\text{Drinker}, \text{Beer}) & \quad \text{Sells}(\text{Bar}, \text{Beer}, \text{Amount})
\end{align*}
\]

a) Write an SQL query (or view) for the relation \( \text{Friends}(F_1, F_2) \) that contains a pair of drinkers \((F_1, F_2)\) iff \( F_2 \) frequents in at least one bar that \( F_1 \) also frequents and that bar serves one or more beers they both like.

b) Given the definition in (a), the answer relation \( \text{Friends}(F_1, F_2) \) is symmetric (i.e., \( \text{Friends}(F_1, F_2) \) implies \( \text{Friends}(F_2, F_1) \)), so every pair \( \{F_1, F_2\} \) is reported twice. How could you obtain only one tuple with \( F_1 \) and \( F_2 \)? Describe what is being defined by this new relation \( \text{Friends}'(F_1, F_2) \).

c) Write relational algebra expressions for (a): Consider

- the “direct” (naive) translation from SQL (i.e., where SELECT-FROM-WHERE is modeled as \( \pi_{\ldots}(\sigma_{\ldots}(R_1 \times \cdots \times R_n)) \)), and
- an expression using \( \Theta \)-joins and atomic join conditions.

d) Find the bar that sells the highest total amount of beer. Hint: you can use a nested query in the HAVING clause of an SQL query.

\[
\begin{align*}
\text{friends}(F1,F2) & \leftarrow \\
& F1 !\neq F2, \\
& \text{frequents}(F1,\text{Bar}), \text{frequents}(F2,\text{Bar}), \\
& \text{serves}(\text{Bar}, \text{Beer}), \\
& \text{likes}(F1,\text{Beer}), \text{likes}(F2,\text{Beer}).
\end{align*}
\]

SELECT F1.Drinker, F2.Drinker
FROM Frequents F1 F2, Serves S, Likes L1 L2
WHERE F1.Drinker != F2.Drinker
AND F1.Bar = F2.Bar \% frequent same bar
AND L1.Drinker = F1.Drinker \% talking about the same guys
AND L2.Drinker = F2.Drinker
AND S.Bar = F1.Bar  % same bar
AND S.Beer = L1.Beer  % same beer

project([F1.Drinker, F2.Drinker],
    select(<CONDITION ABOVE>,
        product(Frequents F1, Frequents F2,
            Serves S,
            Likes L1, Likes L2)))

***alternative solutions:
-- different equalities
-- nested queries
-- joins:

j1 = select([F1.drinker!=F2.drinker],
    join([F1.bar=F2.bar], Frequents F1, Frequents F2))

j2 = join([L1.beer=L2.beer], Likes L1, Likes L2)

j3 = join([L1.drinker=F1.drinker, L2.drinker=F2.drinker], j1, j2)

j4 = join([S.bar=F1.bar, S.beer=L1], j3, Serves S)

answer = project([F1.drinker, F2.drinker], j4)

SELECT Bar FROM Sells
GROUP BY Bar
HAVING SUM(Amount) >= ALL
    (SELECT SUM(Amount) FROM Sells GROUP By Bar)

Exercise 2 (SQL, Universal Statements) Given the schema Emp(Name, Salary),
define in SQL the query “find all employees with the highest salary”:

- using ALL
  
  SELECT E1.Name FROM Emp E1
  WHERE Salary >= ALL
      (SELECT E2. Salary FROM Emp E2)

- using aggregation with MAX
  
  SELECT E1.Name FROM Emp E1
  WHERE E1.Salary IN
      (SELECT MAX (E2.Salary) FROM Emp E2)

- using neither of the above.


```
SELECT E1.Name FROM Emp E1
WHERE E1.Name NOT IN
(SELECT E2.Name FROM Emp E2 E3
```

**Exercise 3 (Indexes)** Explain the different types of indexes:

- **primary vs. secondary indexes**
  
  A primary index determines the location of the indexed information, e.g., a file sorted on the search key index, hash tables determining the bucket. (Another definition is: An index that includes the primary key is called a primary index.) A non-primary index is a secondary index. A secondary index should be dense, since unlike the primary index— the file organisation doesn’t support finding the records of non-indexed key values.

- **dense vs. sparse indexes**

  an index is dense if all values of the indexed attribute appear in it, else sparse (typically one entry for each page aka block of records in the data file).

- **multi-level indexes**

  an index on an index— used e.g., when the first-level index is huge and thus needs to be indexed itself

- **multi-dimensional indexes**

  the search key is a vector of attributes, e.g., spatial information with (X,Y) coordinates

**Exercise 4 (Indexes)** Provide reasons when we should not keep an index on an attribute A.

- if the index is not useful for the typical queries (e.g., don’t index on A, if there are no selections or joins on A); indexes need to be maintained for database updates (insertions, deletions); indexes require storage space

**Exercise 5 (Extensible Hashing)** Consider an extensible hash structure with buckets holding up to three records. Initially the structure is empty. Then, the following records are inserted in the given order a, · · · j (the hashed key is shown in brackets):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>f[010111]</td>
<td>g[101001]</td>
<td>h[011010]</td>
<td>i[011010]</td>
<td>j[001110]</td>
</tr>
</tbody>
</table>

a) Show the structure after these records have been inserted.

- **ins a, b, c, d, e, f; depth=1:**
  
  \[ b(0) = (a,b,d), b(1) = (c,e,f) \]

- **ins g; depth=2:**
  
  \[ b(0) = (a,b,d), b(01) = (e,g), b(11) = (c,f) \]

- **ins h, i:**
  
  \[ b(00) = (b,d), b(10) = (a,h,i), b(01) = (e,g), b(11) = (c,f) \]

- **ins j; depth=3:**
  
  \[ b(000) = (b,d), b(010) = (h,i), b(110) = (a,j), b(011) = (e,g), b(111) = (c,f) \]
Exercise 6 (B-Trees) Consider a B+ tree of order 4, which is initially empty. Show a snapshot of the tree after each of the following insertions have been done (depict the inserted key and the corresponding pointer):

a) insert where key K=1
b) next, insert keys 2, 3, 4, and 5
c) then 6, 7, ... 14.

basic principle: insert into leaf if room; otherwise split leaf; at the level above this looks like an insert.

When inserting into node N the 5th (in general n+1st) key, we create a new node M. The first 3 key-pointer pairs remain in N, the other ones go into M; insert pointer to M and leftmost key of M into the parent.

(1)... (1,2,3) 4 (4,5).
(1,2,3) 4 (4,5,6) 7 (7,8,9) 10 (10,11,12) 13 (13,14)

Exercise 7 (B-Trees) What is the minimum number of record pointers that a B+ tree of order n can contain, given that it has j levels?

assuming that on each level the branching factor is at least n/2, we get (n/2)j

Exercise 8 (Rewriting Rules) Are the following equivalences valid? Give a proof using known other equivalences/rewritings, or give a counter example. We assume set semantics.

a) \( \pi_A(R_1 - R_2) = \pi_A(R_1) - \pi_A(R_2) \)
   no: \( R_1(a,b), R_2(a,c) \)

b) \( S \bowtie_{A=B} (R_1 \cup R_2) = (S \bowtie_{A=B} R_1) \cup (S \bowtie_{A=B} R_2) \)
   \( S \bowtie_{A=B} (R_1 \cup R_2) \)
   = \( \sigma_{A=B}(S \times (R_1 \cup R_2)) \)
   = \( \sigma_{A=B}(S \times R_1 \cup S \times R_2) \)
   = \( \sigma_{A=B}(S \times R_1) \cup \sigma_{A=B}(S \times R_2) \)
   = \( (S \bowtie_{A=B} R_1) \cup (S \bowtie_{A=B} R_2) \)

   \( R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T \)

c) \( R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T \)

Exercise 9 (Rewritings) Consider the following query

\[
\text{SELECT * FROM } R, S, T \\
\text{WHERE } R.A=S.A \text{ AND } R.B=T.B \text{ AND } S.C=T.C \text{ AND } R.D=16 \text{ AND } S.F=17
\]

a) Give different algebraic expressions that contain no “∗”, have pushed \( \sigma \) down, and have only atomic join conditions.

consider schema: \( R(A,B,D,...), S(A,C,F,...), T(B,C,...) \); on top of each of the joins \( R \bowtie S, R \bowtie T, S \bowtie T \) there is another join with the remaining relation. The selections can be pushed down to \( R \) and \( S \) respectively.