Problem 1 (SQL, 20 (2,7,11)) Consider the following relations $\text{Country}(\text{Co\_Name}, \text{Population})$ and $\text{City}(\text{Ci\_Name}, \text{Co\_name}, \text{Population})$. For example, here is an excerpt:

<table>
<thead>
<tr>
<th>Country</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co_Name</td>
<td>Population</td>
</tr>
<tr>
<td>Germany</td>
<td>83536115</td>
</tr>
<tr>
<td>USA</td>
<td>266476278</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Express in SQL the following queries:

a) "Find all countries that have a city ‘Victoria’.”

```sql
SELECT Co\_Name
FROM City
WHERE Ci\_Name = 'Victoria';
```

Now that was easy! Let’s assume a more realistic schema, where City has a foreign key Co\_Id (the id of a country) which refers to the primary key Co\_Id of Country (this is what the problem should have been…). Then the solutions are:

```sql
SELECT Co.Co\_Name
FROM Country Co, City Ci
WHERE Ci.Ci\_Name = 'Victoria'
AND Ci.Co\_Id = Co.Co\_Id;

Alternative solution with a nested query:

```sql
SELECT Co\_Name
FROM Country
WHERE Co\_Id IN
(SELECT Co\_Id FROM City
WHERE Ci\_Name = 'Victoria');
```
b) “Find all countries $C$ and the sum $S$ of all of $C$’s city populations, provided $S$ is at least 10,000,000.”

```sql
SELECT Co_Name, SUM(Population) FROM City GROUP BY Co_Name HAVING SUM(Population) >= 10000000;
```

c) “Find all countries where all cities have at most 5,000,000 inhabitants.”

```sql
SELECT Co_Name FROM City /* or Country */ WHERE Co_Name NOT IN (SELECT Co_Name FROM City WHERE Population > 5000000)
alternative solution:

SELECT Co_Name FROM City GROUP BY Co_Name HAVING MAX(Population) <= 5000000
```
Problem 2 (Algebra & Rewritings, 20) Consider the following SQL query over the schema $R(A,B,\ldots)$ and $S(A,C,\ldots)$:

$$Q: \text{SELECT R.B}$$
$$\text{FROM } R, S$$
$$\text{WHERE } R.A = S.A$$
$$\text{AND } S.C = 'foo';$$

a) Give an equivalent algebra expression for $Q$ using the “direct” translation of SQL, i.e., where SELECT-FROM-WHERE is modeled as $\pi_{\ldots}(\sigma_{\ldots}(R_1 \times \cdots \times R_n))$,

$$\pi_B(\sigma_{R.A=S.A \land C='foo'}(R \times S))$$

b) Same as (a) but give an “optimized” expression, i.e., using join, selections pushed down, and atomic conditions,

$$\pi_B(R \bowtie_{R.A=S.A} \pi_A(\sigma_{C='foo'}(S)))$$

c) Assume the first line of $Q$ above is “SELECT R.A”. How can you further optimize your expression from (b)?

$$\pi_A(R \bowtie_{R.A=S.A} \pi_A(\sigma_{C='foo'}(S)))$$

or $\pi_A(\pi_A(R) \bowtie_{R.A=S.A} \pi_A(\sigma_{C='foo'}(S)))$ which is essentially $\pi_A(R) \cap \pi_A(\sigma_{C='foo'}(S))$

d) Rewrite the expression from (a) into the form given in (b) using equivalences from the class.

$$\begin{align*}
\pi_B(\sigma_{R.A=S.A \land C='foo'}(R \times S)) \\
= \pi_B(\sigma_{R.A=S.A}(\sigma_{C='foo'}(R \times S))) & \quad \text{(decompose \land)} \\
= \pi_B(\sigma_{R.A=S.A}(R \times \sigma_{C='foo'}(S))) & \quad \text{(push selection)} \\
= \pi_B(R \bowtie_{R.A=S.A}(\sigma_{C='foo'}(S))) & \quad \text{(replace } \sigma(\cdot) \text{ by } \bowtie(\cdot))
\end{align*}$$
Problem 3 (Indexes, 10) Indicate whether the following statements are true or not and give a brief explanation or counterexample.

a) A secondary index should be sparse.

No. A secondary index has to be dense since the file is not organized according to a secondary index. Otherwise, the index structure does not allow to find all values directly.

b) A second level index should be sparse.

Yes. If it were dense, we would have the same number of entries in the second level as in the first level index which defeats the purpose of the second level index.

c) In query optimization, it is always better to perform projections as early as possible.

No. There may be an index on the stored relation whereas after the projection, there may be no index. Of course we consider only correct applications of π, i.e., which do not eliminate attributes needed later. But this was not the point here. (Otherwise we could say pushing σ is not always good since we could push it incorrectly.)

d) For queries of the form \(\sigma_{A=a}(R)\) hashing is usually better than a B+ tree.

yes, since we can directly access the record

e) For queries of the form \(\sigma_{A>a}(R)\) hashing is usually better than a B+ tree.

no, hashing does not support range queries since a hash function needs a specific key value (whereas we don’t know the values of A given just \(A > a\))
Problem 4 (Extensible Hashing, 10) 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):

\[
\begin{align*}
a & [101001] \\
b & [010111] \\
c & [001110] \\
d & [011010] \\
e & [101001] \\
f & [011010] \\
g & [010000] \\
h & [111100] \\
i & [010111] \\
j & [000110] \\
\end{align*}
\]

- Show the structure after each directory doubling step and the final structure.

starting with the LSB (from the right):

depth=1; ins a,b,c,d,e,f:

\[
b(0) = (c,d,f), \quad b(1) = (a,b,e)
\]

ins g \sim split b(0); depth=2; ins h:

\[
b(00) = (g,h), \quad b(01) = (c,d,f), \quad b(1) = (a,b,e)
\]

ins i \sim split b(1):

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

ins j \sim split b(10); depth=3:

\[
b(000) = (g,h), \quad b(001) = (d,f), \quad b(010) = (c,j), \quad b(011) = (a,e), \quad b(10) = (b,i)
\]

starting with the MSB (from the left):

depth=1; ins a,b,c,d,e:

\[
b(0) = (b,c,d), \quad b(1) = (a,e)
\]

ins f \sim split b(0); depth=2:

\[
b(00) = (c), \quad b(01) = (b,d,f), \quad b(1) = (a,e)
\]

ins g \sim split b(01); depth=3; ins h,i,j:

\[
b(000) = (c,j), \quad b(001) = (b,j,i), \quad b(011) = (d,f), \quad b(10) = (a,e,h)
\]

5
Problem 5 (B+ Trees, 10) Consider a B+ tree of order 4, which is initially empty. Show some intermediate steps and the final tree, when records with the following keys are inserted (in this order):

- 10; 7; 8; 9; 11; 6; 5; 12; 15; 13

\[
\begin{align*}
\text{ins } 10;7;8;9 &\leadsto (7,8,9,10) \\
\text{ins } 11 &\leadsto (7,8,9) \ 10 \ (10,11) \\
\text{ins } 6 &\leadsto (6,7,8,9) \ 10 \ (10,11) \\
\text{ins } 5 &\leadsto (5,6,7) \ 8 \ (8,9) \ 10 \ (10,11) \\
\text{ins } 12,15 &\leadsto (5,6,7) \ 8 \ (8,9) \ 10 \ (10,11,12,15) \\
\text{ins } 13 &\leadsto (5,6,7) \ 8 \ (8,9) \ 10 \ (10,11,12) \ 13 \ (13,15)
\end{align*}
\]
Problem 6 (B+ Trees, 10) Consider B+ trees of order 2, i.e., where each node holds either one or two key values. Give an example of a 3-level B+ tree that can be reorganized into a 2-level B+ tree with exactly the same values. Display both the 3-level and the 2-level tree. Use letters or numbers for the key values.

For example:

Order two, three levels

```
    50
   / \    \
  30 50   70
 / \   / \   / \   / \   / \    \
20 40 60 80 20 30 40 50 60 70 80
10 20 30 40 50 60 70 80
```

Order two, two levels

```
   40 70
  /    /
20 30 50 60 80
 / \   / \   / \   / \   / \    \
10 20 30 40 50 60 70 80
```

Make sure no B-tree properties are violated
(e.g., n keys, n+1 pointers and ORDER wrt. descendants!)