A: The Yelp Database [70pts]

Suppose we are given a database with the following schema.

**Users** (UserID INTEGER, Name CHAR(30), Age INTEGER, ReviewCount INTEGER)

**Businesses** (BusinessID INTEGER, BName CHAR(30), City CHAR(20), State CHAR(2))

**Checkins** (BusinessID INTEGER, Weekdays INTEGER, Weekends INTEGER)

**Reviews** (ReviewID INTEGER, UserID INTEGER, BusinessID INTEGER, Stars REAL)

**Reviews** (UserID) is a foreign key referring to **Users** (UserID).
**Reviews** (BusinessID) is a foreign key referring to **Businesses** (BusinessID).
**Checkins** (BusinessID) is a foreign key referring to **Businesses** (BusinessID).

A page is 8 kB in size. The RDBMS buffer pool has 10,000 pages, all of which are usable. Initially, the buffer pool is empty.

The relation instances have the following statistics. Assume there are no NULL values. Each integer or real is 8B, and each character is 1B (so as an example CHAR(20) is 20B). Additionally, the record id of each tuple is 8B.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Number of Pages</th>
<th>Number of Tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>75,684</td>
<td>10m</td>
</tr>
<tr>
<td>Businesses</td>
<td>41,504</td>
<td>5m</td>
</tr>
<tr>
<td>Checkins</td>
<td>19,532</td>
<td>5m</td>
</tr>
<tr>
<td>Reviews</td>
<td>488,282</td>
<td>100m</td>
</tr>
</tbody>
</table>

Answer the following questions. Clearly explain how you obtained your answer for each.

1. [10pts] Name 5 indexes (hash and/or clustered B+ tree) on Users that match the predicate in the following SQL query and explain why each index matches.

   ```sql
   SELECT *
   FROM Users
   WHERE NOT ((Name <> "John" AND NOT (Name = "Mary"))
   OR (Age <> 20 AND Age <= 50));
   ```

   ```sql
   SELECT *
   FROM Users
   WHERE Name <> "John" AND Name = "Mary"
   OR Age <> 20 AND Age <= 50;
   ```
**Answer:** Predicate rewritten in CNF is \((\text{Name} = "John" \text{ OR Name} = "Mary") \text{ AND} (\text{Age} = 20 \text{ OR Age} > 50)\). Thus, the following indexes match (reasons are obvious; check your class notes):

- Hash index on Name
- Clustered B+ tree index on Name
- Clustered B+ tree index on (Name, . . . )
- Clustered B+ tree index on Age
- Clustered B+ tree index on (Age, . . . )

2. **[10pts]** Suppose we are given a clustered B+ tree index on Businesses (State, City) with a fan-out of 100. Also, suppose that the index follows the alternative of storing the data records directly in the leaf pages of the index. What is the I/O cost (number of page I/Os) of the following SQL query? Exclude the cost of writing the output. Assume that the selectivity of the predicate \(\text{State} = "WI"\) is 2%. Show all of your calculations clearly.

\[
\text{SELECT DISTINCT City}
\text{FROM Businesses}
\text{WHERE State = "WI";}
\]

**Answer:** Totally, 834 page I/Os. This is a simple Select-Project query, and we use the index to get the selection done. We need \(\lceil \log_{100}(41,504) \rceil = 3\) page I/Os to get to leaf level (recall that the fan-out, which is the number of children pointers in each index node, determines the height of the tree). After that, we need \(\lceil 41,504 \times 0.02 \rceil = 831\) page I/Os at leaf level to get all pages that match the predicate (pages are already sorted on State). Finally, we can do a simple in-memory hash-based (or sort-based) Project since the whole set of selected leaf pages easily fits in memory.

3. **[15pts]** Suppose we are given a clustered B+ tree index each on Businesses (BusinessID) and Checkins (BusinessID), both with a fan-out of 100. Also, suppose that both indexes follow the alternative of storing the data records directly in the leaf pages of the index. Which join algorithm among the following has the lowest I/O cost for a natural join of Businesses and Checkins: Block Nested Loop Join, Index Nested Loop Join, Index Nest, Sort-Merge Join, or Hash Join? Show all of your calculations clearly.

**Answer:** It is a key-key join and both tables are already sorted on BusinessID. Thus, the lowest I/O cost is for Sort-Merge Join, which simply scans both tables in their existing order. The exact costs are listed below:

- \(\text{SMJ: 61,036} = 41,504 + 19,532\). This is already the lower bound!
- \(\text{INLJ: Between 71,137 (41,504 + 19,532 + 1 + 100 + 10,000)}\) and \(\text{119,632 (41,504 + 19,532 \times 4)}\) depending on the buffer replacement policy
4. [10pts] Suppose that there is no index on the Businesses relation. Consider the following SQL query.

```sql
SELECT City, COUNT (BusinessID)
FROM Businesses
GROUP BY City;
```

What is the maximum number of cities for which it is possible to implement hash-based aggregation by reading the relation only once? Assume that the fudge factor of the hash table is $f = 1.4$. Show all of your calculations clearly.

**Answer:** Each entry in the hash table will be of size $20B + 8B = 28B$ ($20B$ for the key, and $8B$ for the count, which is an integer). If $C$ is the number of cities, the size of the hash table (in pages) will be $f \times C \times 28 / (8 \times 1024)$. This must fit in the $B - 1$ available buffer pages. Thus, we have $C \leq 8 \times 1024 \times (B - 1) / (28 \times f)$, which means $C \leq 2,089,586$.

5. [10pts] Suppose that there are no indexes on any relation and no relation is sorted on any attribute. Propose a physical plan for the following SQL query that does not materialize any intermediate relation, and compute its I/O cost. Assume that the values of Stars are real numbers uniformly distributed between 0 and 5 (inclusive), and the values of Age are integers uniformly distributed between 10 and 99 (inclusive). Also assume independence of the predicates on Stars and Age. Show all of your calculations clearly.

```sql
SELECT COUNT (UserID)
FROM Users U, Reviews R
WHERE U.UserID = R.UserID AND R.Stars < 1 AND U.Age = 18;
```

**Answer:** A simple physical plan is to first perform a hash join between Users and Reviews, then pipeline the result to the selection, and finally pipeline the result of the selection to an aggregation operator (which is just a count). The cost of the plan is just the cost of the hash join. The partition definitely fits in memory (since $B$ is much much bigger compared to $\sqrt{1.4 \times 75,684}$), so the I/O cost will be $3 \times (75,684 + 488,282) = 1,691,898$.

6. [15pts] Suppose we are given a hash index on Users (Age), and suppose that the index follows the alternative of storing the data records on a different clustered file. Propose a physical plan for the following SQL query and compute its I/O cost. Your plan must use the available hash index, and include at every operator
whether it uses materialization or pipelining. The same assumptions as the previous question hold for the distribution of the values of Stars and Age. Also assume the independence of the two predicates. Show all of your calculations clearly.

```
SELECT COUNT (DISTINCT UserID)
FROM Users U, Reviews R
WHERE U.UserID = R.UserID AND R.Stars < 1 AND U.Age = 18;
```

**Answer:** The proposed physical plan first applies the selection Age = 18 on Users using the hash index and materializes the result to disk. The selectivity of the selection is $1/(99 - 10 + 1)$, and so, the output size will be $[75,684/90] = 841$ pages. Thus, the I/O cost is $1 + 2 \times 841 = 1,683$. The plan also applies the selection Stars < 1 to Reviews and materializes the result. The selectivity of this selection is $1/5 = 0.2$, and so, its output size will be $[488,282 \times 0.2] = 97,657$. Thus, the I/O cost of this is $488,282 + 97,657 = 585,939$.

The next step in the plan is to join the two selection outputs using a hash-join algorithm. The size of a hash table on the materialized subset of Users is only $[f \times 841] = 1,178$, which would easily fit in buffer memory fully. Thus, no partitioning is needed. Thus, the I/O cost of this is simply $97,657 + 841 = 98,498$.

In order to compute the number of tuples in the intermediate relation that is the output of the join, observe that the join is a key-foreign key join and we are given independence of the selection predicates. Thus, the number of tuples is $100m \times 0.2 \times 1/90 \approx 222,223$.

The next step in the plan is to do a hash-based projection that eliminates all other attributes other than UserID and builds a hash table on UserID for deduplication. The maximum possible size of this hash table is only $[f \times 222,223 \times 8B/8kB] = 304$ pages, which would also easily fit in memory. Thus, this hash-based projection can be pipelined with the hash-join. Finally, the COUNT aggregation on the output of the projection can also be pipelined. Thus, no further I/Os are needed after the join!

Overall, the total cost is: $1,683 + 585,939 + 98,498 = 686,120$ page I/Os.

---

**B: MISCELLANEOUS [30pts]**

1. **[15pts] Hash Join with non-uniform partitioning:**

   Suppose we are joining two tables S and R with respective number of pages $4BN_S$ and $8BN_R$, wherein $4BN_S \gg 8BN_R$. The number of buffer pages is $4B + 1$ and the buffer pool is initially empty. We are also given that $2fN_R = 4B - 1$, where $f$ is the hash table fudge factor.
The distribution of the join attribute values in $S$ and $R$ are such that after the first hash partitioning phase, we get exactly $4B$ partitions of $S$, each of length $N_S$ pages, but not all partitions of $R$ are of the same length. Suppose $R$ gets partitioned as follows: $2B$ partitions of length $N_R$ pages, $B$ partitions of length $2N_R$ pages, and $B$ partitions of length $4N_R$ pages.

What is the I/O cost of the regular hash join algorithm discussed in class? Exclude the cost of writing the output of the join. Assume perfect uniform splitting occurs during the recursive repartitioning. Show all of your calculations clearly.

(Hint: The answer is of the following form: $xBN_S + yBN_R$, where $x \in \{12, 14, 16, 18\}$ and $y \in \{24, 28, 32, 36\}$.)

**Answer:** The total I/O cost is $14BN_S + 32BN_R$. The lower bound is $3 \times (4BN_S + 8BN_R) = 12BN_S + 24BN_R$ (regular hash-join cost). But we need to recursively repartition the $B$ partitions of $R$ that are of length $4N_R$ pages each, which means we also need to repartition the corresponding $B$ partitions of $S$ that are of length $N_S$ pages each. Thus, the I/O cost goes up by $2 \times B \times 4N_R + 2 \times B \times N_S = 2BN_S + 8BN_R$.

2. **[15pts] Query Rewriting:**

Consider the following query expressed in Relational Algebra over the Yelp database given in Part A:

$$
\pi_{BName}(\sigma_{\text{Stars} > 4, \text{ReviewCount} \geq 100}(\text{Users} \bowtie \text{Reviews} \bowtie \text{Businesses}))
$$

Write an equivalent Relational Algebra expression where the selections and projections are pushed down the plan as far as possible. Apply only the rules from Chapter 15.3.4 of the Cow Book. Show and explain each step in your rewriting.

**Answer:** The equivalent RA expression is as follows:

$$
\pi_{BName}(\pi_{BusinessID}(\pi_{UserID}(\sigma_{ReviewCount \geq 100}(\text{Users}))
\bowtie \pi_{UserID,BusinessID}(\sigma_{\text{Stars} > 4}(\text{Reviews}))
\bowtie \pi_{BName,BusinessID}(\text{Businesses}))
$$

The above rewriting is a straightforward application of the following six rules: cascading of $\pi$, cascading of $\sigma$, commutativity of $\sigma$, commuting $\sigma$ and $\pi$, "pushing the select" through $\bowtie$, and commuting $\pi$ and $\bowtie$. 
1. [10pts] Consider the Yelp database given in Part A. What is the I/O cost of bulk loading a clustered B+ tree index on Reviews (Stars)? Assume that we follow the alternative of storing the data records directly in the leaf pages of the index. Also, assume that all non-leaf pages are kept in buffer memory during the bulk loading process, but are all written out to disk once at the end of the process. Include this cost of persisting the index. Show all of your calculations clearly.

Answer: Bulk loading = External merge sort (EMS) + scan + write out index.

The cost of EMS is: 
\[2 \times 488,282 \times (1 + \lceil \log_{9,999}(\lceil 488,282/10,000 \rceil) \rceil) = 1,953,128.\]

Notice that even if replacement sort and/or double buffering are used, the I/O cost is the same!

Suppose the fan-out of the index is 100 (this was given elsewhere, but our assumption is to use that here too). The number of pages at the highest index level is \(\lceil 488,282/100 \rceil = 4,883\) (this is the maximum; it could be lower if there are duplicates in Stars values). The next lower level has \(\lceil 4,883/100 \rceil = 49\) pages, and finally, there is 1 root page. Thus, the (maximum) size of the index is 4,933 pages.

Overall, the total I/O cost is 1,953,128 + 488,282 + 4,933 = 2,446,343.

The fan-out can be larger. Since the index nodes only store (Stars, PageID) pairs (of size 16B), the fan-out can be as large as 8 \(\times\) 1024/16 = 512. If we use this fan-out, the number of nodes in the index levels are \(\lceil 488,282/512 \rceil = 954\), \(\lceil 954/512 \rceil = 2\), and 1, which adds up to an index size of only 957 pages. This yields a marginally smaller total I/O cost of 1,953,128 + 488,282 + 957 = 2,442,367.

2. [10pts] Let \(R, S, T\) be three relations such that \(S\) and \(T\) have the same schema. Consider the following equation:

\[R \bowtie_{\theta} (S - T) = (R \bowtie_{\theta} S) - (R \bowtie_{\theta} T)\]

where \(\theta\) is a join condition. Is the above identity true? If yes, provide a proof. If no, give a counterexample that falsifies the identity.

Answer: Yes, they are equivalent. To prove this, we will show that \(R \bowtie_{\theta} (S - T) \subseteq (R \bowtie_{\theta} S) - (R \bowtie_{\theta} T)\) and \(R \bowtie_{\theta} (S - T) \supseteq (R \bowtie_{\theta} S) - (R \bowtie_{\theta} T)\).

For the one direction, suppose a tuple \(t \in R \bowtie_{\theta} (S - T)\). Then, there exists a tuple \(t_R \in R\), and a tuple \(t_S \in S\) that joins according to the \(\theta\)-condition. Also, \(t_S \notin T\). This means that \(t \in R \bowtie_{\theta} S\), and also \(t \notin R \bowtie_{\theta} T\). So, we have that \(t \in (R \bowtie_{\theta} S) - (R \bowtie_{\theta} T)\).

For the reverse direction, suppose a tuple \(t \in (R \bowtie_{\theta} S) - (R \bowtie_{\theta} T)\). Then, \(t \in (R \bowtie_{\theta} S)\), so there must be a tuple \(t_R \in R\) and a tuple \(t_S \in S\) that join according to the \(\theta\)-condition. The tuple \(t_S\) can not belong in \(T\), since \(t \notin (R \bowtie_{\theta} T)\). So \(t_S \in S - T\), which implies that \(t \in R \bowtie_{\theta} (S - T)\).