Q 1. [20pts] For the following questions, clearly circle True or False.

1. All secondary indexes are unclustered indexes.
   False

2. Median is not an algebraic aggregate.
   True

3. Rename is the only operator in relational algebra whose runtime does not vary across different instances of the same relation schema.
   True

4. It is impossible for a parallel RDBMS to exhibit super-linear speedups.
   False

5. Natural joins are always equi joins.
   True

6. The blocked I/O improvement to the external merge sort may increase the I/O cost by increasing the number of merge passes.
   True

7. The iterator interface for physical operators is the main software abstraction used to implement pipelining.
   True

8. Sort-merge join is never amenable to pipelining and is always a blocking operator.
   False
9. All recoverable schedules are Avoid-Cascading-Abort (ACA) schedules.
   False

10. The union operator can always be executed using just an unmodified hash join im-
    plementation.
    False

Q 2. [24pts] For the following questions, clearly circle the right answer. Only one option
    is correct.

1. Which relational operator implementation in a parallel RDBMS never needs on-the-
    fly re-partitioning of the data?

   (a) ×                      (b) ❧                    (c) π               (d) σ               (e) γ

   ANSWER: (d)

2. Which index alternative stores the records themselves in the data entries?

   (a) AltRID              (b) AltRecord          (c) AltRIDList       (d) None of the above

   ANSWER: (b)

3. Which ACID property states that the changes of committed transactions will be per-
   sisted in the database?

   (a) Atomicity          (b) Consistency       (c) Isolation        (d) Durability

   ANSWER: (d)

4. Which of the following paradigms of parallelism is often called “symmetric multi-
   processing” (SMP) database systems?

   (a) Shared-disk          (b) Shared-memory     (c) Shared-nothing

   ANSWER: (b)
5. Which join order enumerations are the most amenable to pipelining if BNLJ was the only physical join operator allowed?

(a) Left deep trees  (b) Right deep trees  (c) Bushy trees  (d) All of the above

ANSWER: (a)

6. Which of the following invariants/conditions always hold for an extendible hash index? GD and LD stand for global and local depths respectively.

(a) All LD > GD  (b) All LD ≥ GD  (c) All LD < GD  (d) All LD ≤ GD

ANSWER: (d)

7. Which of these SQL capabilities has no counterpart in (extended) relational algebra?

(a) WHERE  (b) GROUP BY  (c) LIMIT  (d) UNION  (e) SELECT DISTINCT

ANSWER: (c)

8. Which improvement to the external merge sort was designed to exploit the direct-memory access (DMA) capability of modern machines?

(a) Internal replacement sort  (b) Double buffering  (c) Blocked I/O

ANSWER: (b)

9. In which set operator’s hashing-based implementation does the in-memory hash table hold (a part of) the output?

(a) ∪  (b) ∩  (c) −  (d) ×

ANSWER: (a)

10. Which SQL isolation level ensures that the phantom problem will not arise?

(a) Read Uncommitted  (b) Read Committed  (c) Repeatable Read  (d) Serializable

ANSWER: (d)
11. Which component of the access latency of a magnetic hard disk is affected by how fast the head moves?

(a) Rotational delay  (b) Seek time  (c) Transfer time  (d) None of the above

ANSWER: (b)

12. Which is the most common data partitioning scheme for ETL in parallel RDBMSs?

(a) Round Robin  (b) Range-based  (c) Hashing-based  (d) Random

ANSWER: (c)

Q 3. [36pts] Consider the simplified sales database schema given to you in the separate sheet. The arrows represent key-foreign key dependencies.

1. (3pts) Write an SQL query to compute the total sales in California stores in 2019.

   ANSWER:
   
   ```sql
   SELECT SUM (Price * Quantity) FROM Sales, Items, Stores
   WHERE Sales.ItemID = Items.ItemID
   AND Sales.StoreID = Stores.StoreID
   AND Sales.Year = 2019 AND Stores.S_State = 'California'
   ```

2. (3pts) Write a relational algebra expression to answer the same query as in the previous question. You can draw the logical query plan instead if you wish.

   ANSWER:
   
   ```sql
   γSUM(Price∗Quantity)(σS_State='California'(Stores ◦◁ (σYear=2019(Sales ◦◁ Items)))
   ```
   
   Other right answers possible.

3. (3pts) Write a different relational algebra expression than your response to the above question that is logically equivalent and also answers the query in the first question. Once again, you can draw the logical query plan if you wish. (Hint: Use the algebraic rewrite rules covered in class.)

   ANSWER: Many right answers possible.

4. (4pts) Draw a physical query plan corresponding to one of your above two logical query plans. Mention which one it corresponds to. Clearly show which physical operator implementation was picked for each logical operator.

   ANSWER: Many right answers possible.
5. (3pts) Name 3 indexes, with at least one being a hash index, that match the selections in the query given in the first question (total sales in California stores in 2019).

**ANSWER:** Hash index on Sales.Year or Stores.S_State. Clustered B+ tree index on Sales.Year or Stores.S_State or any other index key with either of these are prefixes.

6. (9pts) For the following pairs of queries, indicate if they are logically equivalent.

   (a) $Q_1 : \gamma_{\text{Supplier}, \text{AVG}(\text{Price})}(\text{Items})$  
      $Q_2 : \gamma_{\text{Supplier}, \text{AVG}(\text{Price})}(\pi_{\text{Supplier}, \text{Price}}(\text{Items}))$

   **No.** Deduplication on $(\text{Supplier}, \text{Price})$ in $Q_2$ could reduce the counts for some groups.

   (b) $Q_1 : (\text{Sales} \bowtie \text{Stores}) \bowtie \text{Customers}$  
      $Q_2 : (\text{Stores} \times \text{Customers}) \bowtie \text{Sales}$

   **Yes.** In $Q_2$, although the crossproduct introduces new tuples, the natural join with Sales will filter them away implicitly. This is because this natural join is essentially a conjunctive equi-join predicate on Customers.CustomerID and Stores.StoreID. Thus, it will always yield the same output as $Q_1$.

   (c) $Q_1 : \sigma_{\text{Price} \geq 100}(\text{Items}) \bowtie \text{Sales} - \sigma_{(\text{Price} \geq 100 \land \text{Month} = \text{December})}(\text{Sales} \bowtie \text{Items})$

   $Q_2 : \sigma_{(\text{Price} \geq 100 \land \text{Month} \neq \text{December})}(\text{Sales} \bowtie \text{Items})$

   **Yes.** December and non-December months are mutually exclusive.

7. (11pts) For the following questions, use these pieces of information about the given sales database and the machine environment:

   $N_{\text{Sales}} = 10^8$, $N_{\text{Stores}} = 10^5$, $N_{\text{Items}} = 10^6$, and $N_{\text{Customers}} = 10^7$ (respective number of pages of each table in row store format)
   Page size is 8 KB. Available buffer memory is 32 GB.
   The fudge factor for hash tables is 1.4.
   The size of each attribute is 8 B, except for S_State, I_Name, Supplier, Category, C_Name, and C_State, each which is 40 B.

   Assume there are no indexes and the buffer pool is empty to start with. Exclude the cost of writing the output of a given query in your I/O cost calculations.
(a) (3pts) Which of the following joins cannot be executed with an I/O cost that involves just one read of each base table?

(i) $\text{Sales} \bowtie \text{Customers}$  
(ii) $\text{Sales} \bowtie \text{Stores}$  
(iii) $\text{Sales} \bowtie \text{Items}$

**Answer:** (i). The hash table on Customers, the smaller table, is too big to fit in RAM. So, a partition phase is needed.

(b) (3pts) What is the I/O cost of the fastest possible physical query plan for the following query?

$$\sigma_{\text{CustomerID}=123}((\text{Customers} \bowtie \text{Sales})?$$

(i) $10^7$  
(ii) $10^8$  
(iii) $11 \cdot 10^7$  
(iv) $12 \cdot 10^7$  
(v) $33 \cdot 10^7$

**Answer:** (iii). Push the select on CustomerID through the join. Because CustomerID arises in both tables, we can filter both on the fly as we scan them. Also, CustomerID is the primary key in Customers; so, it will yield at most one tuple there. Thus, the whole query only needs one read of each base table.

(c) (5pts) What is the I/O cost of the fastest possible physical query plan for the following query?

$$\gamma \text{COUNT(DISTINCT CustomerID)}(\sigma_{\text{C\_State}\neq\text{S\_State}}((\text{Customers} \bowtie \text{Sales}) \bowtie \text{Stores}))?$$

(i) $1101 \cdot 10^5$  
(ii) $1303 \cdot 10^5$  
(iii) $3003 \cdot 10^5$  
(iv) $3301 \cdot 10^5$

(v) $3303 \cdot 10^5$  
(vi) $1505 \cdot 10^5$  
(vii) $5105 \cdot 10^5$  
(viii) $5501 \cdot 10^5$

**Answer:** (iv). The costliest operation is $\text{Customers} \bowtie \text{Sales}$. BNLJ would require 4 passes over Sales, which is more expensive than HJ or SMJ, both of which will have the cost of $3 \cdot (N_{\text{Customers}} + N_{\text{Sales}})$ due to the amount of RAM given. This cost is unavoidable. The goal is to pick the physical operators such that we do not exceed this cost significantly. We note that the Stores table is small enough to fit in RAM entirely as a hash table. But note that we need to count distinct CustomerID values in the final downstream operator, which requires an implicit project on CustomerID. We could consider a hash-based or sort-based project. If we choose SMJ for the first join, this downstream project operator will receive its input in an already sorted order, which helps us avoid extra cost.
Thus, overall, the physical query plan we pick is as follows: filescans feeding into SMJ for Customers $\bowtie$ Sales, followed by a downstream HJ with Stores as inner table and the SMJ’s output as outer table (note that no partitioning is needed for this HJ), followed by a pipelined select to filter on the given inequality predicate, followed by a pipelined project on CustomerID, which is finally followed by a pipelined counting. The total I/O cost of this plan is $3 \cdot (N_{Customers} + N_{Sales}) + N_{Stores} = 3301 \cdot 10^5$.

Q 4. [20pts] You are given a database with three distinct data objects A, B, and C. You are also given the following three transactions that arrive concurrently.

$T1 : R(A), W(B), Commit$
$T2 : R(B), W(A), Commit$
$T3 : W(A), W(B), Commit$

Consider the following three interleaved schedules.

$S1 : R_{T1}(A), W_{T1}(B), Commit_{T1}, R_{T2}(B), W_{T2}(A), Commit_{T2}, W_{T3}(A), W_{T3}(B), Commit_{T3}$

$S2 : R_{T1}(A), R_{T2}(B), W_{T1}(B), W_{T2}(A), Commit_{T1}, Commit_{T2}, W_{T3}(A), W_{T3}(B), Commit_{T3}$

$S3 : W_{T3}(A), R_{T1}(A), W_{T1}(B), R_{T2}(B), W_{T2}(A), W_{T3}(B), Commit_{T1}, Commit_{T2}, Commit_{T3}$

$S4 : R_{T1}(A), W_{T3}(A), W_{T1}(B), Commit_{T1}, W_{T3}(B), Commit_{T3}, R_{T2}(B), W_{T2}(A), Commit_{T2}$

Answer the following questions by clearly circling all correct options and only the correct options. Multiple options may be correct. If none of the given options are correct, clearly circle only the “None” option.

If multiple options are correct but you circle only a subset of the correct options, you will get points proportionally. But if you circle any wrong options, you will lose points proportionally! But the score for each question is lower bounded by zero.

1. (2pts) Which schedules are serial?

   S1    S2    S3    S4    None

Answer: Only S1.
2. (4pts) Which schedules are serializable?

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>None</th>
</tr>
</thead>
</table>

**Answer:** S1 and S4. S4 is equivalent to T1 → T3 → T2. S3 is clearly not serializable due to numerous conflicts. But S2 is also not serializable because its first part is not equivalent to either T1 → T2 or T2 → T1 (see reason below); note that T3 clearly starts only after both T1 and T2 finish. So, overall, S2 is not equivalent to any of the 6 possible serial schedules.

3. (4pts) Which schedules are recoverable?

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>None</th>
</tr>
</thead>
</table>

**Answer:** S1, S2, and S4. None of these have dirty reads. But in S3, T1 reads A dirtied by T3 and commits first.

4. (3pts) Which schedules have no conflicts?

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>None</th>
</tr>
</thead>
</table>

**Answer:** Only S1. S3 clearly has WW conflicts on A and B. But S2 and S4 have subtle RW conflicts although there are no “repeated” reads per se. In S2, the write of A by T2 might implicitly depend on the value of B read by T2. But T1 came in between to write B after T2 read B but before T2 commits. Similarly the other way around for the write of B by T1. These cyclic RW conflicts are why S2 is not serializable. However, in S4, T3 writes A after T1 read it but before T1 commits. Interestingly, although S4 has this single RW conflict, it is still serializable and recoverable.

5. (3pts) Which schedules have a WW conflict?

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>None</th>
</tr>
</thead>
</table>

**Answer:** Only S3.
6. (4pts) Suppose we use the Repeatable Read isolation level of SQL. Which schedules will lead to a deadlock?

- S1
- S2
- S3
- S4
- None

**Answer:** S2 and S3.

**Extra Credit Q. [5pts].** Consider the same sales database used for Q3. Use the same database statistics and machine setup as Q3.7. What is the rough I/O cost (in number of pages) of the following query? Ignore record-level metadata overheads. Explain your calculation briefly.

\[ \pi_{\text{CustomerID}}(\text{Sales}) \]

Now suppose we use a column store format instead of row store. What is the rough I/O cost (in number of pages) of the above query now? Explain your calculation briefly. One again, ignore record-level metadata overheads. Assume no compression is used.

**Answer:** The temporary file’s size is \( N_T = \frac{1}{8} N_{\text{Sales}} = 12.5 \text{ million pages} \). So, the buffer memory is likely not enough for a one-pass project but is enough for a two-phase hash-based project. The I/O cost becomes \( N_{\text{Sales}} + 2N_T = 125 \text{ million pages} \).

With a column store, we only need to read the CustomerID column. We already know its size is \( N_T = 12.5 \text{ million pages} \). For the hash-based project, it suffices to read just this for partitioning. So, the I/O cost is now only \( N_T + 2N_T = 37.5 \text{ million pages} \). So, a column store could make this query about 3x faster!