Q 1. [40pts] For each question below, select the right option (only one is correct).

1. Which of the following relational operators can be processed using a regular (unmodified) hash join implementation?

   (a) Union  (b) Intersection  (c) Set difference  (d) Select  (e) Project

   ANSWER: (b)

2. Which of the following SQL aggregates requires a shuffle among worker nodes in a parallel DBMS even when the GROUP BY list is empty?

   (a) SUM  (b) AVG  (c) VARIANCE  (d) MEDIAN  (e) MAX

   ANSWER: (d). Recall from class that MEDIAN is not an algebraic aggregate.

3. Which file organization is typically the most efficient for inserting new records?

   (a) Heap file  (b) Sorted file  (c) B+ tree index with AltRecord

   ANSWER: (a)

4. We are given this join query: $R \bowtie S \bowtie T \bowtie U$. Recall that some query optimizers only consider left-deep join trees for join order enumeration. How many different left-deep join trees exist for this query? (Hint: Swapping the left and right input of a $\bowtie$ in a given tree yields a different tree.)

   (a) 4  (b) 6  (c) 15  (d) 16  (e) 24

   ANSWER: (e). The left-deep tree has 4 spots; so, the number of trees is the number of permutations of those 4 spots, i.e., $4! = 24$.

5. Which is the dominant parallelism paradigm that is used in parallel DBMSs, MapReduce/Hadoop, and Spark?

   (a) Shared-nothing  (b) Shared-memory  (c) Shared-Disk
ANSWER: (a)

6. Which of the following relational operators does not preserve the schema of (at least one of) their inputs?

(a) Union (b) Intersection (c) Set difference (d) Select (e) Project

ANSWER: (e)

7. In a hard disk, which of the following components of the data access time accounts for the delay caused by the radial movement of the arm?

(a) Seek time (b) Rotational delay (c) Transfer time

ANSWER: (a)

8. In a B+ tree index, which nodes are allowed to have duplicates of the index key?

(a) Leaf nodes (b) Root node (c) Non-root internal nodes

ANSWER: (a)

9. Which of the following symbols does not represent a relational operator from the extended relational algebra?

(a) γ (b) ∪ (c) π (d) μ (e) ×

ANSWER: (d)

10. Which data system introduced the first major SQL-on-Hadoop dialect?

(a) Pig (b) Hive (c) Spark (d) Polybase (e) None of the rest

ANSWER: (b)

11. Which of the following EMS optimizations cannot raise the I/O cost?

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

ANSWER: (a)
12. In an extendible hash index with GD 5, how many pointers from the directory will a bucket with LD 2 have?

(a) 1  (b) 2  (c) 4  (d) 8  (e) None of the rest

ANSWER: (d)

13. A primary index is necessarily also the following type of index?

(a) Unique  (b) Secondary  (c) Composite  (d) AltRID  (e) B+ tree

ANSWER: (a)

14. Which join order is most amenable to pipelining when using only hash joins?

(a) Left deep tree  (b) Right deep tree  (c) Bushy tree  (d) All of them

ANSWER: (b)

15. A given query on a parallel RDBMS takes 20min to finish on a 4-worker cluster. If the RDBMS exhibits linear speedup for this query, what will its runtime be on a 20-worker cluster?

(a) 20min  (b) 10min  (c) 5min  (d) 4min  (e) None of the rest

ANSWER: (d)

16. Which of these SQL capabilities does not have a counterpart in extended relational algebra?

(a) SELECT DISTINCT  (b) WHERE  (c) JOIN  (d) GROUP BY  (e) ORDER BY

ANSWER: (e)

17. What is the minimum number of distinct hash functions needed for the two-phase improved hash join implementation in a parallel RDBMS?

(a) 1  (b) 2  (c) 3  (d) 4  (e) 5

ANSWER: (c)
18. Which page format does not have metadata for each record slot?

(a) Packed  (b) Unpacked  (c) Variable-length  (d) None of the rest

ANSWER: (a)

19. Which equi-join implementation’s I/O cost is hard to accurately express in closed analytical form?

(a) BNLJ  (b) INLJ  (c) SMJ  (d) HJ  (e) None of the rest

ANSWER: (b)

20. Which of the following machine resources will likely face dramatically higher loads in the serverless and resource-disaggregated design of cloud computing relative to previous generations of cloud designs?

(a) Processor  (b) Memory  (c) Disk  (d) Network  (e) None of the rest

ANSWER: (d)

Q 2. [10pts] Hash Join with non-uniform partitioning. You are joining two tables R and S, which have $4BN_R$ and $12BN_S$ pages respectively, using a hash join. The number of available buffer pages is $4B + 1$. The buffer pool is initially empty. You are given that $4BN_R \gg 12BN_S$ and $2FN_S = 4B - 1$, where $F$ is the hash table fudge factor.

The distribution of the join attribute values in R and S are such that after the first hash partitioning phase, we get exactly $4B$ partitions each of R and S. Each partition of R has $N_R$ pages, but the partitions of S have differing sizes. Suppose S gets partitioned as follows: $B$ partitions of size $N_S$ pages each, $2B$ partitions of size $3N_S$ pages each, and $B$ partitions of size $5N_S$ pages each.

What is the I/O cost of the above join using the regular hash join algorithm discussed in class? Exclude the cost of writing the output. Assume that perfect uniform splitting occurs during the recursive repartitioning and that we do not need to recurse more than once. Briefly explain and show all of your calculations clearly.

(Hint: The answer is of the following form: $xBN_R + yBN_S$, where $x \in \{10, 12, 14, 16, 18, 20\}$ and $y \in \{50, 54, 58, 62, 66, 70\}$.)

Answer: The total I/O cost is $18BN_R + 58BN_S$.

Clearly, the hash table is built on S, since it is the smaller relation. The lower bound is $3 \times (4BN_R + 12BN_S) = 12BN_R + 36BN_S$ (regular hash join cost).
But we need to recursively repartition the $2B + B$ partitions of S that are larger than $2N_S$ pages each, since the given buffer memory is not enough to fit their hash tables fully. This means we also need to repartition the corresponding $3B$ partitions of R that are of length $N_R$ pages each.

Thus, the I/O cost goes up by $2 \times (2B \times 3N_S + B \times 5N_S + 3B \times N_R) = 6BN_R + 22BN_S$.

Q 3. [10pts] **Buffer replacement.** You are given an initially empty buffer pool with 3 buffer frames. There are 5 pages on disk: A, B, C, D, E. The buffer replacement policy used is LRU. Answer the following questions for the following sequence of page operations being executed. There are no partial credits for each sub-question.


1. (4pts) Circle all the pages (and only the pages) that are present in the buffer pool at the end of the sequence.

   **Answer:** A, C, and E. For replacement with LRU, the frame with dirty B gets evicted with a flush. Second request for C is a cache hit.

2. (3pts) How many page flushes take place? Circle only your final answer.

   **Answer:** 1. To flush B.

3. (3pts) What is the total page I/O cost (reads and writes)? Circle only your final answer.

   **Answer:** 5. Breakdown: 4 reads and 1 flush write.

Q 4. [10pts] **MapReduce.** Consider this Netflix database table used in class: Ratings ($RID$, MovieID, UserID, Stars, RateDate). Suppose it is stored as a large distributed CSV file on HDFS hash partitioned on RID. You are asked to answer the following query: Get the number of five-star ratings for each movie in the table.

1. (7pts) Write a MapReduce job to answer the query. Clearly explain the Map and Reduce functions separately. Pseudocode or just precise prose suffices.

   **Answer:** Map: For loop through the tuples in a shard; check if Stars is 5; if so, emit (MovieID, 1) as key-value pair for that tuple; if not, skip that tuple.

   Reduce: Iterator has list of 1 values for a unique MovieID; for loop to add them up; emit MovieID and final sum as the number of five-star ratings.
2. (3pts) Suppose you have Hive or SparkSQL installed. Write a single SQL statement to answer the same query. Your query must be fully correct; no partial credits.

**Answer:** SELECT MovieID, COUNT(*) FROM Ratings WHERE Stars = 5 GROUP BY MovieID;

---

Q 5. [15pts] **Logical query optimization.** Are you on social media? Given the following simplified relational database schema for social networks.

*Person (ID, Name, Age)*

*Friends (ID1, ID2)*

Person.ID is the primary key of that table. Friends.ID1 and Friends.ID2 are foreign keys referring to Person.ID.

For each query given, clearly circle which query listed under it is logically equivalent to it? For each question only one option is correct. No partial credits.

1. (4pts) \( \pi_{\text{Name}}(\sigma_{\text{Age} > 20}(\text{Person})) \)

   (a) \( \sigma_{\text{Age} > 20}(\pi_{\text{Name}}(\text{Person})) \)

   (b) \( \pi_{\text{Name,ID}}(\sigma_{\text{Age} > 20}(\text{Person})) \)

   (c) \( \pi_{\text{Name}}(\sigma_{\text{Age} > 20}(\pi_{\text{Name,Age}}(\text{Person}))) \)

   (d) \( \sigma_{\text{Age} > 20}(\pi_{\text{Name,ID}}(\text{Person})) \)

   (e) \( \sigma_{\text{Name}}(\pi_{\text{Age} > 20}(\text{Person})) \)

   (f) \( \pi_{\text{Name}}(\pi_{\text{Age}}(\sigma_{\text{Age} > 20}(\text{Person}))) \)
(g) None of the above

Answer: (c)

2. (5pts) \( \sigma_{(Age>20) \land (ID2=1234) \land (Name='Thanos') \land (ID=ID1)} (Person \times Friends) \)

(a) \( \sigma_{(Age>20) \land (Name='Thanos')} (Person) \times \sigma_{(ID2=1234) \land (ID=ID1)} (Friends) \)

(b) \( \sigma_{ID=ID1} (\sigma_{ID=1234} (Friends)) \times \sigma_{(Name='Thanos') \land (Age>20)} (Person) \)

(c) \( \sigma_{(Age>20) \land (Name='Thanos')} (Person) \bowtie_{ID=ID1} \sigma_{ID1=1234} (Friends) \)

(d) \( \sigma_{ID2=1234} (Friends) \bowtie_{ID=ID1} \sigma_{(Age>20) \land (Name='Thanos')} (Person) \)

(e) None of the above

Answer: (d)

3. (6pts) \( \pi_{Name,ID} (Person \bowtie_{ID=ID2} \pi_{ID2} (\sigma_{Name='Thanos'} (Person \bowtie_{ID=ID1} Friends))) \)

(a) \( \pi_{Name} (\pi_{ID1,ID2} (\pi_{ID} (\sigma_{Name='Thanos'} (Person)) \bowtie_{ID=ID1} Friends) \bowtie Person) \)

(b) \( \pi_{Name,ID} (Person) \bowtie_{ID=ID2} \pi_{ID2} (\sigma_{Name='Thanos'} (Person) \bowtie_{ID=ID1} Friends) \)

(c) \( \pi_{Name,ID} (Person) \bowtie_{ID=ID1} \pi_{ID1} (\sigma_{Name='Thanos'} (Person) \bowtie_{ID=ID1} Friends) \)

(d) \( \pi_{Name,ID2} (Person) \bowtie_{ID=ID2} \pi_{ID2} (\sigma_{Name='Thanos'} (Person) \bowtie_{ID=ID2} Friends) \)
(e) None of the above

**Answer: (e).** Note that although (b) comes close to being equivalent, the schemas will not be the same because (b) will have three attributes: *Name, ID, and ID2.*

**Q 6. [15pts] Operator implementation and PQP costing.** Do you like shopping? Consider the following simplified relational database schema for shopping.

*Products (PID, PName, Vendor, Price)*
*Customers (CID, CName, Zipcode, Age)*
*Purchases (CID, PID, Timestamp, Quantity)*

*Products.*PID and *Customers.*CID are primary keys. In *Purchases, CID* and *PID* are foreign keys referring to *Customers.*CID and *Products.*PID, respectively. The size of each attribute is 8 bytes, except for *PName, Vendor,* and *CName,* which are 40 bytes each.

The tables are stored in a row-store heap file format with the following numbers of pages: 1 million for *Products, 10* million for *Customers,* and 1 billion for *Purchases.* No indexes are present in the database. Page size is 8 KB. Available buffer memory is 64 GB. The buffer pool is empty to start with. The fudge factor for hash tables is 1.4.

For each query given, what is the **lowest possible I/O cost** (in pages) using only the physical operator implementations discussed in class? Exclude output write costs. No partial credits for the first two sub-questions. Just circle your final answer.

1. (3pts) *Purchases ⊙◁ Customers.*

**Answer:** Buffer memory available is $64 \text{ GB} / 8 \text{ KB} = 8 \text{ million pages}$. HJ or SMJ would cost $3 \times (10 + 1000) = 3030 \text{ mil}$. BNLJ costs $10 + \lceil (1.4 \times 10\text{mil} / (8\text{mil} - 2)) \rceil \times 1000 = 2010 \text{ mil pages}$. So, BNLJ is the lowest and the answer is **None of the rest**. (NB: Previous version answer had HJ/SMJ and 3030 as the answer, but I raised the buffer memory size and forgot to account for BNLJ afterward.)

2. (4pts) $\pi_{CID,Age} (Purchases \bowtie Customers)$.

**Answer:** Push down projection through the join. Non-deduplicated *CID* and *Age* from *Customers* will be $(16 / 64)$ of the table size, which is 2.5 mil pages. So, the hash table on it fits fully in memory for a hash join (3.5 mil pages). We can also push down *CID* to *Purchases* but that is redundant, since we only need one streaming read of it. I/O cost: $10 + 1000 = 1010 \text{ million pages}$. Note that ignoring the join with *Purchases* is incorrect because some *CID* may not have purchased anything.
3. (8pts) Get the total sales made in each zip code. (For this one, make sure to explain all steps, starting with the relational algebra query or LQP, any logical rewrites used, the final PQP chosen, any pipelining used, and relevant buffer memory calculations. Partial credits are possible.)

**Answer:** First step is to translate the English query to a relational algebra query. Clearly, we need a GROUP BY aggregate to compute the sum of product price and purchase quantity. This requires a full 3-table join. So, the query is:

\[ \gamma_{\text{Zipcode}, \text{SUM}(\text{Price}\times\text{Quantity})}(\text{Products} \bowtie \text{Purchases} \bowtie \text{Customers}) \].

The best case scenario for lowest I/O cost is to answer the query with just a single scan of each table. That can happen with a PQP that is a right deep tree of hash joins in which both hash tables fit together in DRAM, followed by the GROUP BY operator also finishing in a pipelined manner. Buffer memory is 8mil pages; Products is 1mil; Customers is 10mil. So, while the hash table on Products can fit in DRAM, that on Customers will not. Thus, next we need to see if we can reduce the size of the data flowing into the hash join. To this end, we draw upon the idea from Exercise 4 Q6.C (also in the previous sub-question) and apply a logical rewrite to push down the implicit projection in the GROUP BY aggregate down through the joins to the two smallest base tables, viz., Products and Customers.

The final GROUP BY only needs Zipcode from Customers but the join with Purchases needs CID. So, we can insert a non-deduplicating project on \{Zipcode, CID\} on Customers first. The intermediate non-deduplicating project output size is \(\frac{16}{64} = \frac{1}{4}\) size of Customers, which is 2.5mil. Hash table for the join on this intermediate inner table is 3.5mil.

Likewise, the final GROUP BY and the join with Purchases together only need \{PID, Price\} from Products. That intermediate non-deduplicating project output size is \(\frac{16}{96} = \frac{1}{6}\) size of Products, which is 0.17mil. Hash table for the join on this intermediate inner table is 0.24mil.

The last piece left is the final GROUP BY itself. Suppose we choose the hash-based GROUP BY. The hash table’s cardinality, viz., the number of unique Zipcode values, is at most the cardinality of Customers. Also, SUM only needs the partial running sum as the sufficient statistic. So, the upper bound on this hash table’s size is \(1.4 \times \left(\frac{16}{64}\right) \times 10\text{mil} = 3.5\text{mil}\).

Overall, all 3 hash tables (2 for joins, 1 for GROUP BY) all fit in DRAM together (7.24mil < 8mil). This means the entire PQP is fully pipelined, with no repartitioning or intermediate materialization. We only need one streaming read of each base table. This is the lowest possible I/O cost anyway, and it adds up to 1011 million.
Extra Credit Question. [5pts] Suppose we manage to accomplish full resource disaggregation parallelism for a next generation cloud-native RDBMS. Briefly explain why this can be beneficial for reducing query latency vs. the prior art of shared-nothing parallel execution on a provisioned cluster (e.g., a la Redshift) using a realistic query example on the Netflix database discussed in class. Clearly state the query, show its PQP, and explain what resources are elastically used to accelerate which part of the PQP. Assume that the database is too large to fit entirely in DRAM though.

Answer: Many answers possible. One good answer is to construct a query with a select, join, and aggregate operators in the LQP. Then for a PQP with a hash join, elastically scale up the DRAM to ensure the hash join does not need partitioning. Then for the aggregate elastically scale up the CPU cores available to raise computation parallelism.