1 (22 pts) Terms and Definitions

1.1 (10 pts) True or False

(2.5 points each) In the following questions, circle True or False and include a 1 line description of why. **Draw a box around your final answer**.

1. Given a database $\mathcal{D}$, it is possible to develop software to automatically identify the functional dependencies using techniques such as correlation analysis. **True/False**

> False. Functional dependencies are integrity constraints, which must hold for ALL valid instances of the database.

2. A query optimizer such as the Selinger Optimizer finds the query plan with the minimum estimated execution cost. **True/False**

> False. Query optimization is hard and the number of plans is too large to find the best plan. Selinger goes for the least worst plan.

3. A search key is a type of candidate key that is optimized for searching over. **True/False**

> False. The search key, used for indexes, has no relation to the candidate key, a type of functional dependencies.

4. A table exhibits redundancy when 2 or more rows have the same value for 1 or more attributes in the schema. **True/False**

> False. Redundancy is defined with respect to the functional dependencies on a table.

1.2 (12 pts) Short Answers

(3 points each) In *at most* one short sentence each, answer the following questions as they relate to database management systems. **Draw a box around your final answer**.

1. In database recovery, what ACID transaction property is UNDO designed to ensure?

> Atomicity, need to undo inflight transactions.

2. In database recovery, what ACID transaction property is REDO designed to ensure?

> Durability, need to redo committed actions that were not written to disk.

3. Can the following sequence of operations and log writes guarantee correct recovery after a crash at any point in time? Why or Why not?

```
1. BEGIN;
2. Read(A)
3. Write(B)
4. Write(A)
5. flush log record for "Write(A)" to disk
6. flush A to disk
7. flush log record for "Write(B)" to disk
8. flush B to disk
9. flush log record for COMMIT to disk
10. COMMIT;
```

Correctly follows write ahead logging protocol.

4. What is the purpose of a database cursor?

To avoid sending all of the query results to the client, the cursor is an iterator over the results.

2 (14 points) SQL

Consider the following database schema containing actors, movies, and which actor acted in which movie:

```
CREATE TABLE actors(
  aid  int PRIMARY KEY,
  is_tall bool NOT NULL,
  name text UNIQUE NOT NULL
);

CREATE TABLE movies(
  mid  int PRIMARY KEY,
  name text NOT NULL,
  year int NOT NULL
);

CREATE TABLE actedin(
  actor_id int REFERENCES actors(aid),
  movie_id int REFERENCES movies(mid),
  PRIMARY KEY (actor_id, movie_id)
);
```

Write the SQL statements to answer the following statements. You may use temporary tables. **Draw a box around your final answer, that is what we will grade!**

1. (6 pts) Find the names of the movies whose cast contains all tall actors. For example, if there are a total of 5 tall actors, and movie M's cast has 10 actors, and all 5 tall actors are in M's cast, then M should be in the result set.

This is classic division:

```
with tall as (SELECT * FROM actors WHERE is_tall = true)
SELECT M.name
FROM movies M
```

```
WHERE NOT EXISTS ( SELECT T.aid
                   FROM tall T
                   WHERE T.aid NOT IN (SELECT AI.actor_id
                                       FROM actedin AI
                                       WHERE AI.movie_id = M.mid));
```

Or exotic group by

```
SELECT M.name
FROM movies M, actedin AI, actors A
WHERE M.mid = AI.movie_id AND
      A.aid = AI.actor_id AND
      A.is_tall = true
GROUP BY M.name
HAVING count(distinct A.aid) = (SELECT count(*)
                                FROM Actors A2
                                WHERE A2.is_tall = true);
```

4 points for correct division or group by structure. 2 points if fully correct

2. (8 pts) Find the ids of all unique pairs of actors that have acted together in two different movies where at least one of the actors is tall. We consider (A, B) to be the same as (B, A), so if (A, B) is in the result, then (B, A) should NOT be in the result.

There are a few ways to solve this:

```
SELECT distinct AI1.actor_id, AI3.actor_id
FROM actors A1, actors A3, actedin AI1, actedin AI2,
     actedin AI3, actedin AI4
WHERE A1.aid = AI1.actor_id AND
      A3.aid = AI3.actor_id AND
      (a1.is_tall OR a3.is_tall) AND
      AI1.actor_id = AI2.actor_id AND
      AI3.actor_id = AI4.actor_id AND
      AI1.movie_id = AI3.movie_id AND
      AI2.movie_id = AI4.movie_id AND
      AI1.movie_id <> AI2.movie_id AND
      AI1.actor_id < AI3.actor_id;
```

OR

```
SELECT AI1.actor_id, AI2.actor_id
FROM actors A1, actors A2, actedin AI1, actedin AI2
WHERE A1.aid = AI1.actor_id AND
```

```
        A2.aid = AI2.actor_id AND
        (A1.is_tall OR A2.is_tall) AND
        AI1.movie_id = AI2.movie_id AND
        AI1.actor_id < AI2.actor_id
GROUP BY AI1.actor_id, AI2.actor_id
HAVING COUNT(distinct AI1.movie_id) >= 2;
```

+4 points if join structure is correct, +2 points if inequality(ies) is correct, full points if fully correct e.g., distinct in the first case (ignoring typose)

3 (14 points) Functional Dependencies

Consider the table `ABCDEF`, where each letter represents an attribute in the schema. Let the table have the following functional dependencies

$$A \rightarrow BCDEF \tag{1}$$
$$B \rightarrow CD \tag{2}$$
$$E \rightarrow DF \tag{3}$$
$$FB \rightarrow D \tag{4}$$

Draw a box around your final answer – that is what we will grade!

1. (6 pts) Write one (of possibly many) BCNF decompositions of the table `ABCDEF` given the functional dependencies above.

```
ABE, EF, BCD

or

ABE, BC, EDF

or

ABE, BC, EF, FB, BD
```

2. (2 pts) ***Briefly*** describe whether the decomposition exhibits any anomalies, and if so, list them. (This answer is contingent on a correct decomposition above).

The decomposition is not dependency preserving.

3. (6 pts) Write the minimal cover of the above set of functional dependencies.

```
A->B
A->E
B->C
B->D
E->D
E->F
```

+1 for each correct FD above. -2 if extra or incorrect FDs.

4 (28 pts) Statistics and Access Paths

Given the tables `S(sid, name, age)` and `T(tid, topic)`, where there are secondary B+Tree indexes on `S(sid)` and `T(tid)`, and a secondary B+Tree index on `S(age)`. Also consider the following statistics about the tables.

```
NCARD(S)      = 500
ICARD(S.sid)  = 100
ICARD(S.age)  = 50
minmax(S.age) = [0, 50)   -- from 0 to 49, inclusive

NCARD(T)      = 1000
ICARD(T.tid)  = 100

Page size     = 1020 bytes
Pointer size  = 10 bytes
Key size      = 10 bytes
Tuple size    = 50 bytes  -- both S and T tuples have same size
Fill Factor   = 0.5
default selectivity: 0.1
```

Assume that leaf pages store (key, pointer) pairs. Assume that a tuple is fully stored within its page (and not split across multiple pages). The following questions refer to the query below:

```
SELECT *
FROM S JOIN T ON S.sid = T.tid
WHERE age < 5
```

Draw a box around your answer, that is what we will grade. Clearly list your assumptions, if any.

Some useful statistics given the above information:

- Each (key, pointer) pair is 20 bytes large. Each page only uses 510 bytes to store data.
- Each directory page stores N keys and $N+1$ pointers, where $N = 25$ based on $10(N+1)+10*N = 510$,
- Each leaf page stores 25 (key, pointer) entries.
- Each data page that stores tuples contains $\frac{510}{50} = 10$ tuples.

1. (3 pts) What is the selectivity of the `age < 5` predicate?

(5 / 50) = 0.1

2. (3 pts) What is the expected cardinality of the query?

$$\text{join selectivity} = \frac{1}{\max(\text{keys in S.sid}, \text{keys in T.tid})} = \frac{1}{\max(100, 50)} = \frac{1}{100} = 0.01$$
$$\begin{aligned}\text{selectivity} &= \text{NCARD(S)} \times \text{NCARD(T)} \times (\text{join selectivity}) \times (\text{select selectivity}) \\ &= 500 \times 1000 \times 0.01 \times 0.1 = 500\end{aligned}$$

3. (6 pts) What is the best access method for S while taking into account the predicate `age < 5`? How many disk IOs does it cost?

Heap Scan is best: 50 disk IO.

The output of this predicate is 50 tuples. A secondary index would need at least 50 IOs, plus the cost for the initial index lookup and to leaf the leaf pages, so a secondary index will be worse.

Not required, but the full calculation for using the secondary index: a secondary index on S.age needs to store 500 (key, pointer) pairs, which takes $\frac{500}{25} = 20$ pages. This requires a single root node with 26 pointers. To find the starting point for the scan (the minimum value node), we need to read the root (1 I/O), then read the matching leaf pages. There are $\frac{50}{25} = 2$ leaf pages with matching values (2 I/O). This is a total of 3 I/Os to get 50 pointers, plus 50 I/Os to get the tuples, for a total of 53 I/Os.

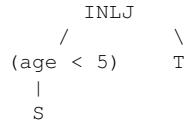
2 pts for correct access method. 2 pts for correct cost estimate.

4. (16 pts) The following questions ask you to estimate the cost of different physical query plans (note, this is not Selinger's `bestjoin()` estimate, but the normal plan cost estimates). We assume the following as part of the cost estimates:

   - If the selection operator is on the outer table, it is pipelined to the parent join operator.
   - If the selection operator is on the inner table, we assume that the selection operator is executed to completion and its results are stored in a temporary table. The temporary table is *then* used as the inner table in the join. In this case, take into account the cost of writing the temporary to disk, and reading it in the parent operator.

   Draw a box around your final cost estimate. Clearly state any assumptions that you make.

   (a) (8 pts) Compute the cost estimate of the following index nested loops join plan:



The cost to read the outer relation for the index nested loops join is given in part 3. The number of tuples from the outer relation is given in part 1. Now we need to compute the index cost, so we need the B+Tree height for the secondary index on T.tid:

$$\#\text{ leaf pages} = \frac{\text{total tuples in T}}{\text{tuples per page}} = \frac{1000}{25} = 40$$
$$\text{index height} = log_{\text{fanout}}(\text{leaf pages}) = log_{26}40 = 1.132 \text{ round up to } 2$$

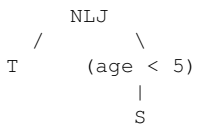
The other way to compute the index height without needing a logarithm is to compute it from the bottom up. If there is a height of 1, that means there is only a root node, which has 26 pointers. Therefore, that can only represent 1 to 26 leaf pages. If you have more than 26, you need another level, which can store a maximum of $26 \times 26 = 676$ leaf pages. Therefore, we need two levels.

Therefore, for each value of T.sid, we need to read the root node plus one directory page (because of the height of 2). We then need 1 I/O to read the leaf page. This gives us 10 matching (key, pointer) pairs. (if there were more than 25, we would need to read more leaf pages.) To get the actual tuples, we need 10 additional I/Os to read the pages in the unordered heap file that contain the tuples. That is 13 I/Os total.

Therefore the total cost is:

$$\begin{aligned}\text{total} &= \text{cost of outer} + (\text{tuples in outer}) \times (\text{cost for index lookup}) \\ &= 50 + 50 \times 13 = 700\end{aligned}$$

(b) (8 pts) Compute the cost estimate of the following nested loops join plan:



Selection cost and temp table: 50 IOs to read S, 5 pages to write output relation.

Join cost: 100 IOs for T + 1000 * 5 = 5100.

Total cost: 50 + 5 + 5100 = 5155

5 (20 points) Query Optimization and Transactions

5.1 (8 pts) Selinger Optimization

This question involves the Selinger optimizer discussed in lecture. We want to join tables A, B, and C, and want to determine the optimal join order. There is a secondary B+ tree index on B, and a secondary B+tree index on C.

Assume Selinger has partially run and determined the following optimal join orderings, where NL = Nested Loops, INL = Index Nested Loops, SM = Sort Merge.

```
bestjoin(A,B) = (A NL B)
bestjoin(B,C) = (B NL C)
bestjoin(A,C) = (A INL C)
```

(2 pts each) Circle true or false for each of the following questions. Include a brief one sentence explanation.

1 pt for T/F, 1 pt for explanation.

1. The Selinger Optimizer would consider the plan `(C NL (A NL B))`. (True / False)

False, not left deep

2. The Selinger Optimizer would consider the plan `((B NL C) INL A)`. (True / False)

False, No index on A

3. The Selinger Optimizer would consider the plan `((B NL A) NL C)`. (True / False)

False, B NL A is not the best join plan for A,B

4. The Selinger Optimizer would consider the plan `((A INL C) INL B)`. (True / False)

True, A INL C is the best join for A,C, and B has an index.

5.2 (12 pts) Transaction Schedules

(1 pt per correct statement) Consider the following transactions schedules, along with statements about the schedule. Draw a circle around each true statement.

1.
```
T1: R(A)       W(B)              R(C)
T2:       W(A)       R(C)  W(B)       R(C) W(C)
```

   (a) The schedule is serial.

False

   (b) The schedule is serializable.

True

   (c) The schedule is conflict serializable.

True

(d) The schedule exhibits the Dirty Read anomaly.

False

(e) The schedule exhibits the Unrepeatable Read anomaly.

False

(f) The schedule exhibits the Lost Writes anomaly.

True

2.
```
T1:             r(A)       w(B)
T2: r(B) w(A)                    w(B)
T3:                  r(A)            w(B)
```

(a) The schedule is serial.

False

(b) The schedule is serializable.

True: This is view serializable. T2's w(B) happens after T1's w(B). However, since it is never read, and it is then overwitten, then it effectively can be reordered before T1's w(B) without affecting the results. The serial order is T2, T1, T3.

(c) The schedule is conflict serializable.

False: T2 writes A before T1 reads it, but it also writes B after T1 writes it.

3.
```
T1:        R(A)        W(B)                    R(C)
T2: W(A)         R(C)        W(B) R(C) W(C)
```

(a) The schedule exhibits the Dirty Read anomaly.

True

(b) The schedule exhibits the Unrepeatable Read anomaly.

False

(c) The schedule exhibits the Lost Writes anomaly.

True

6 (Extra credit)

1. (3.5 points) Draw something here, or don't. Whatever you want. You'll get the points regardless.

A drawing of a dinosaur olleying over the grand canyon. While drinking Capri-Sun. On the moon.