There are 13 questions and 11 pages in this quiz booklet. To receive credit for a question, answer it according to the instructions given. You can receive partial credit on questions. You have 80 minutes to answer the questions.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be harder than others. Attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can’t understand your answer, we can’t give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ. Laptops may be used; no phones or internet allowed.

Do not write in the boxes below

<table>
<thead>
<tr>
<th>1-4 (xx/22)</th>
<th>5-7 (xx/28)</th>
<th>8-10 (xx/22)</th>
<th>11-13 (xx/28)</th>
<th>Total (xx/100)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Name:
I Column Stores

Suppose you are running this query over a database of employees and departments:

SELECT e.rank, d.bldg, AVG(e.sal)
FROM employees e, departments d
WHERE e.dno = d.dno
GROUP BY e.rank, d.bldg

Suppose there are 10,000 employees, 100 departments, each in a distinct building, and 20 ranks, and that salaries are distributed uniformly between $50,000 and $150,000. Assume ranks are uniformly distributed across departments.

Suppose you are evaluating this query in C-Store.

1. **[6 points]**: What is the best sort order for the emp and dept tables, given you are running only the above query? Briefly state why.
   
   emp:
   
   dept:
   
   Why?

2. **[5 points]**: Given the sort orders and table statistics given above, indicate which compression method should be used for each column to make the above query as fast as possible, choosing from:
   
   A. gzip (e.g., Lempel-Ziv compression)
   B. run-length encoding (RLE)
   C. delta encoding
   
   (For each column, indicate your answer (A-E))
   
   - e.rank: ____________
   - d.bldg: ____________
   - e.sal: ____________
   - d.dno: ____________
   - e.dno: ____________

Name:
II Cost Estimates

Suppose you have the following “equi-height” histogram on bank account balances in a database consisting of a table of bank accounts and their balances, as well as customers and their names and addresses:

Each bucket contains 100/6 = 16.7% of the records

3. [6 points]: Consider the predicate “balance > $50,000 and balance < $75,000”. Given the histogram above, what would Postgres estimate the selectivity $F$ of this predicate to be?

(Write your answer in the space below.)

Your database has 10,000 bank accounts in MA. You run the query:

```
SELECT COUNT(*)
FROM accounts, customers
WHERE balance > $50,000 and balance < $75,000
AND customers.state = 'MA'
AND accounts.custid = customers.custid
```

When you run the query, you find the COUNT it produces is just $1 \times 10,000 \times F$, where $F$ is the selectivity Postgres estimated in the previous question.

4. [5 points]:

In one sentence or less, provide one possible reason why the database might produce an answer that is so much less than expected, assuming that the histogram accurately reflects the true distribution of balances in the database:

(Write your answer in the space below.)

Name:
III  Indexing

You have a retail database consisting of a set of orders, each placed at one store, and each consisting of products. The table lineitem stores the list of products sold in each order, including the price paid per item and the quantity of each item purchased.

Consider the following query over this database:

```sql
SELECT orders.storeid, SUM(price * quantity)
FROM lineitem, orders, stores
WHERE stores.region = 'EAST'
AND lineitem.productId = 12345
AND lineitem.orderno = orders.orderno
AND orders.storeid = stores.storeid
GROUP BY orders.storeid
```

Here `storeid` is the primary key of `stores` and `orderno` is the primary key of `orders`. Suppose that there are 1,000 distinct products, 100,000 orders, 100 stores, 1,000,000 lineitems and 10 regions, and that products are equally popular, orders have uniform size, and stores are uniformly distributed across regions.

Your system has sufficient memory to store 50,000 records of any table in memory at a time and can read 10,000 records per second from disk. Seeks take 10 ms. Assume CPU costs are small but consider them when breaking ties if I/O costs are identical.

The database has access to the following join algorithms: grace hash, in-memory hash (when applicable), nested loops, sort merge, and index-nested loops. The system does not consider cross products.

5.  [10 points]: Suppose you have no indexes.

   In the query plan below, write the names of the tables (to specify the join order), and your choice of join algorithm for each join. Assume filters are pushed down as far as possible. For nested-loops joins, the leftmost join is the outer relation. For hash joins indicate which relation you would build the hash table on.

   ![Query Plan Diagram]

   Name:
6. [8 points]: What B+Trees would you recommend the system create if this is the only query the system has to run? For each B+Tree, specify the tables/attributes it is on, and whether or not it is clustered. Assume you can have at most one clustered index per table (otherwise there are no constraints on how many indexes you can create.)

7. [10 points]: Given the indexes you created above, label the diagram below with the names of the tables, your choice of access method for each table, and your choice of join algorithm for each join. Assume filters are pushed down as far as possible (and into indexes when possible). For nested-loops joins, the leftmost join is the outer relation. For hash joins indicate which relation you would build the hash table on.

Name:
IV Entity and Schema Design

Suppose you are designing a schema to record information about the intramural (IM) team sports at MIT, e.g., football, basketball, hockey, etc. Your database needs to record the following information:

- For each student, his/her student id (only MIT students for this problem), name, and address. A student may join different teams.
- For each team, its name, captain, ranking, and team members (including the captain).
- For each game, its host team, guest team, date, and score.

8. [10 points]: Draw an entity relationship diagram for this database. Please draw entities as squares, attributes as ovals, and denote relationships as diamonds between pairs of entities. Label each edge with a “1” or an “N” to indicate whether the entity on the other side of the relationship connects to 1 or N entities on this side of the relationship. For example, the following would indicate that each person lives in 1 city, and multiple people live in each city:

```
Person  N  Lives In  1  City
```

Give each entity, relationship, and attribute a name.

(Draw your diagram in the space below)

Name:
9. **[6 points]**: Use your ER diagram to determine a relational schema for this database. For each table, use the form:

   Tablename (field1-name, ..., fieldn-name)

   to denote its schema. If you wish, you can create an additional field for each table to serve as a unique identifier. **Underline** the field(s) that are the primary key of each table.

10. **[6 points]**:
    Suppose every team hosts at most one game a day, which yields the functional dependency (host team, date) → (guest team, score). How would you modify/decompose your schema to account for this dependency?

    **(Write your answer in the space below.)**
V Query Planning

Suppose you are querying the mimic2 database to compare healthcare outcomes of patients who were cared for in a single unit during their ICU stay versus those who switched care units.

You run the following query to count the number of chart events for each class of patients:

```
SELECT count(*), 'moved' AS moved_status
FROM chartevents a, (SELECT icustay_id FROM icustayevents
    WHERE first_careunit <> last_careunit) AS moved
WHERE a.icustay_id = moved.icustay_id
UNION ALL
SELECT count(*), 'not moved' AS moved_status
FROM chartevents b, (SELECT icustay_id FROM icustayevents
    WHERE first_careunit = last_careunit) AS not_moved
WHERE b.icustay_id = not_moved.icustay_id;
```

This produces the following output:

<table>
<thead>
<tr>
<th>count</th>
<th>moved_status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1942840</td>
<td>moved</td>
</tr>
<tr>
<td>7407892</td>
<td>not moved</td>
</tr>
</tbody>
</table>

(2 rows)

Time: 4398.751 ms
To understand why this query took almost 5 seconds, you run EXPLAIN ANALYZE, and get the following output (some of the output has been omitted for clarity):

```
QUERY PLAN
-> Aggregate (cost=392078.93..392078.94 rows=1)
  (actual time=3068.571..3068.571 rows=1)
  -> Hash Join (cost=53.35..368813.61 rows=9306127)
    (actual time=2.735..2895.545 rows=1942840)
    Hash Cond: (a.icustay_id = public.icustayevents.icustay_id)
    -> Seq Scan on chartevents a (cost=0.00..227957.33 rows=9548444)
      (actual time=0.015..1369.092 rows=9548444)
    -> Hash (cost=33.73..33.73 rows=1570)
      (actual time=0.499..0.499 rows=148)
      Buckets: 1024 Batches: 1 Memory Usage: 6kB
      -> Seq Scan on icustayevents (cost=0.00..33.73 rows=1570)
        (actual time=0.021..0.457 rows=148)
        Filter: (first_careunit <> last_careunit)
        Rows Removed by Filter: 148
    -> Aggregate (cost=58817.74..58817.75 rows=1)
      (actual time=3663.863..3663.863 rows=1)
      -> Nested Loop (cost=0.00..58699.19 rows=47420)
        (actual time=0.052..3065.650 rows=7407892)
        -> Seq Scan on icustayevents (cost=0.00..33.73 rows=8)
          (actual time=0.018..1.260 rows=1430)
          Filter: (first_careunit = last_careunit)
          Rows Removed by Filter: 148
        -> Index Only Scan using chartevents_chartevents_o5 on chartevents b
          (cost=0.00..7267.40 rows=6578)
          (actual time=0.011..1.477 rows=5180)
          Index Cond: (icustay_id = public.icustayevents.icustay_id)
          Heap Fetches: 7407892
```

11. [8 points]: Notice that Postgres chooses very different query plans for the two aggregate queries. Using the EXPLAIN ANALYZE output as a guide, in a few sentences qualitatively explain the differences between the access methods and join algorithms Postgres chose for each aggregate query.

(Write your answer in the space below.)

Name:
12. [10 points]: Based on the EXPLAIN output and what you know about query planning, which of the following statements must be true?

(Circle ‘T’ or ‘F’ for each choice.)

<table>
<thead>
<tr>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>Postgres accurately estimates the cardinality of the two joins</td>
</tr>
<tr>
<td>T</td>
<td>Postgres evaluates the “moved” subquery by doing a sequential scan on icustayevents once per chartevent</td>
</tr>
<tr>
<td>T</td>
<td>Postgres uses the chartevents table as the “inner” relation in both joins</td>
</tr>
<tr>
<td>T</td>
<td>The filter condition in the “moved” subquery is more selective (produces fewer rows) than the filter condition in the “not moved” subquery</td>
</tr>
<tr>
<td>T</td>
<td>The index nested loops join in the second aggregate query causes a seek into the chartevents heapfile for each iteration of the loop</td>
</tr>
</tbody>
</table>

Name:
In order to understand how the query would have performed with a different plan, we temporarily disable hash joins and merge joins, causing Postgres to change the plan it uses for the first aggregate query. When we run the query again, we see that it’s over a second faster. EXPLAIN ANALYZE produces the following output (only the first aggregate query that changed is shown):

```
QUERY PLAN
-------------------------------------------------------------------------------------
-> Aggregate (cost=11320242.85..11320242.86 rows=1)
  (actual time=1764.008..1764.009 rows=1)
  -> Nested Loop (cost=0.00..11296977.54 rows=9306127)
    (actual time=0.080..1509.502 rows=1942840)
     -> Seq Scan on icustayevents (cost=0.00..33.73 rows=1570)
      (actual time=0.022..0.765 rows=148)
      Filter: (first_careunit <> last_careunit)
      Rows Removed by Filter: 1430
     -> Index Only Scan using chartevents_chartevents_o5 on chartevents a
      (cost=0.00..7129.73 rows=6578)
      (actual time=0.027..7.420 rows=13127)
      Index Cond: (icustay_id = icustayevents.icustay_id)
      Heap Fetches: 1942840
```

13. **[10 points]** Briefly explain why the index nested loops join in this query plan is faster than the hash join in the previous plan. Give one reason why Postgres probably chose the wrong plan.

   (Write your answer in the space below.)