EECS 339: Introduction to Database Systems

Midterm Exam Solution
February 17, 2015

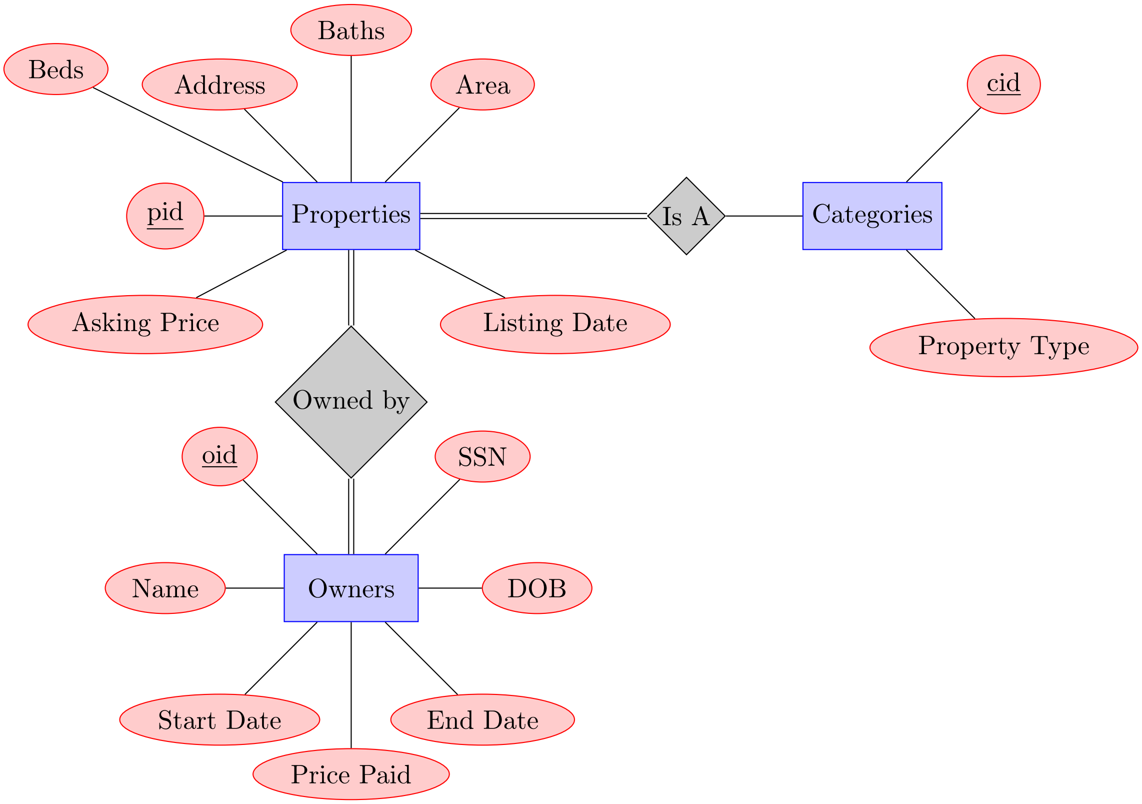
Name:

I Schema Design

1. 5 points: You are designing a real estate database with the following requirements:

- Properties describe houses or plots of land. They have an id, address, an area in square feet, a number of bedrooms, a count of bathrooms, an asking price, and the date it was listed
- A house has an ownership history, where each homeowner has an id, name, date of birth, a social security number, year they bought it, end ownership date, and price paid. An owner may own any number of properties, either in the past or concurrently.
- Properties each have a category, such as commercial, residential, or land. Each distinct category has an id.

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, with a label of the form "1:1", "N:1" or "1:N", where "1:N" indicates that a single entity on the left side has a relationship with many entities on the right, but each entity on the right has a relationship with only one entity on the left. Give each entity, relationship, and attribute a name.



2. 5 points: Using your entity-relationship diagram, create a schema for this database. Use the following format:

```
tablename (attribute1, attribute2, attribute3)
```

Underline the one or more attributes for each primary key.

```
properties (pid, addr, area, beds, baths, ask_price, list_date, cid);
owners (oid, name, dob, ssn, start, end, price, pid);
category (cid, type);
```

3. 5 points: Is this schema in Boyce-Codd Normal Form?

[✓ **YES**] **NO**

Why or why not? If it has any redundancies or anomalies explain them below.

There are no redundancies nor anomalies.

II Query Writing

For the schema below, express the following questions as SQL queries.

4. 15 points: A university's database has the schema:

```
students (sid, s_name, dob)
courses (cid, c_name, year, quarter, meeting_time, building, room)
grades (s_sid, c_cid, grade)
```

(a) In what building did the class with id 'eecs339' meet during the winter quarter of 2015?

```
SELECT building
FROM courses
WHERE cid='eecs339' and quarter='winter' and year=2015;
```

(b) Create a list of each student's id, name, and their GPA.

```
SELECT students.sid, students.s_name,avg(grade)
FROM students, grades
WHERE sid = s_sid GROUP BY s_sid, s_name;
```

(c) List all course ids, course names, and the number of enrolled students. Order the output alphabetically by name.

```
SELECT c_name, count(*)
FROM courses, grades
WHERE cid = c_cid
ORDER by c_name;
```

(d) List the names of all students who got a grade of 4.0 in the course with id 'eecs339'.

```
SELECT s_name
FROM students, courses, grades
WHERE sid = s_sid and cid = c_cid and cid = 'eecs339' and grade=4;
```

(e) What is the `cid` of the class(es) where the students had the highest average grade?

```
with class_avg as (SELECT cid, avg(grade) as avg_grade
    FROM courses, grades
    WHERE cid = c_cid
    GROUP BY cid)
SELECT cid
FROM class_avg
WHERE avg_grade = (
    SELECT max(avg_grade)
    FROM class_avg);
```

III Query Rewriting

5. 10 points: For the following queries, can they be rewritten for faster query execution? If so, rewrite the query. If not, explain why not.

(a)
```
SELECT *
FROM EMPLOYEES
WHERE salary * (1 + 0.05)  > 100000;}
```

Rewrite possible?

[✓ YES] **NO**

Explanation:

```
SELECT *
FROM EMPLOYEES
WHERE salary > 100000/1.05;
```

(b)
```
 WITH ninjas as (SELECT emp_id
    FROM employees as e, departments as d
    WHERE e.dept_no = d.dept_no and d.dept_name = 'ninjas')
SELECT n.name
FROM ninjas as n, favorite_colors as fc
WHERE n.emp_id = fc.emp_id and fc.color = 'black';
```

Rewrite possible?

[✓ YES] NO

Explanation:

```
SELECT e.names
FROM employees as e, departments as d, favorite_colors as fc
WHERE d.dept_name='ninjas' and fc.color = 'black'
AND e.dept_no = d.dept_no AND e.empl_id = fc.empl_id;
```

(c)

```
with feedingCount as (
    SELECT aid,count(*)  as cnt
    FROM feedings
    GROUP BY aid)
SELECT a.aid, a.name, f.cnt
FROM animals as a, feedingCount as f
WHERE f.aid = a.aid AND f.cnt > 2;
```

Rewrite possible?

[✓ YES] NO

Explanation:

```
SELECT a.aid, count(*) as cnt FROM animals feedings WHERE a.aid = f.aid GROUP BY a.aid
HAVING cnt >= 2;
```

(d)

```
with bigTransactions as
     (SELECT  distinct acct_id
      FROM transactions
      WHERE amt > 10000)
SELECT  a.name, a.addr
FROM accounts
WHERE EXISTS (
SELECT * FROM bigTransactions
WHERE bigTransactions.acct_id = accounts.acct_id);
```

Rewrite possible?

[✓ YES] NO

Explanation:

```
SELECT distinct a.name, a.addr FROM accounts as a, transactions as t WHERE a.aid = t.aid
and t.amount > 10000;
```

IV Short Answer

IV.1 Memory Management

6. **5 points:** Based on the DBMIN algorithm from the paper, "An Evaluation of Buffer Management Strategies for Relational Database Systems", what percent of buffer pool accesses will be hits for each of the following

situations create for LRU eviction policies? Assume the system has up to 50 MB of buffer pool space. Each database page is 5 MB.

(a) A 100 MB table with 10 tuples per page that is sequentially scanned.

First tuple in a page is a miss, the remaining reads are hits. $\frac{1}{10}$ misses → 90% hit rate.

(b) A nested loop join with the outer relation having 100 pages and 1,000 tuples total, and the inner relation having 5 pages with 1 tuple per page.

There are 10 tuples per page on the outer relation. Therefore, our accesses look like this:
m m m m m m 40x hits, m h h h, 40x hits

We have 1,000 outer page reads, each compared with 5 inner tuples, producing a total number of $1,000 + 5 * 1,000$ reads. We accumulate misses once per page for our 105 pages. Therefore, we have a hit rate of:
$1 - \frac{105}{6000} = 98.25\%$.

7. 5 points: You have a queue of queries with the following working set sizes (in pages):

13, 16, 11, 18, 20, 5, 2, 15, 7, 3, 10, 1, 8, 14, 4

Your buffer pool contains 30 pages. If the queries all have the same duration, in what order will they be executed?

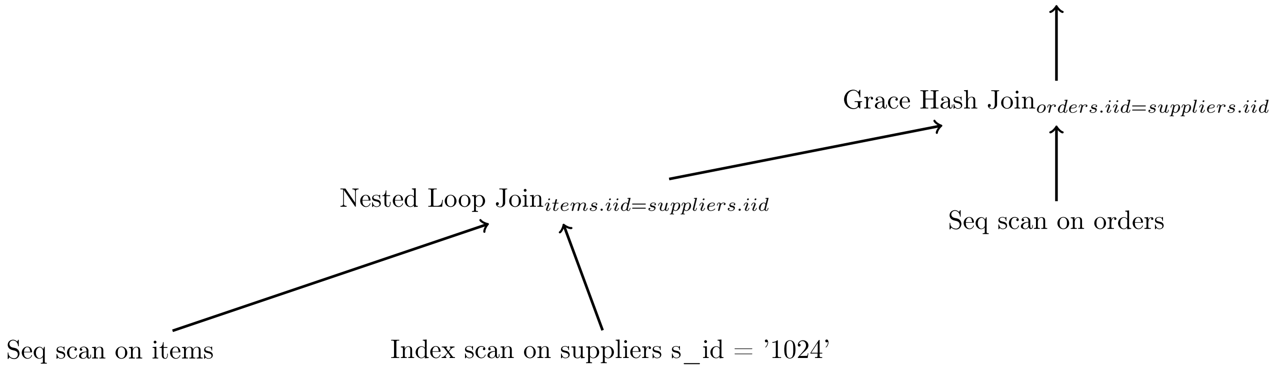
13, 16, 1
11, 18
20, 5, 2, 3
15, 7, 8
10, 14, 4

IV.2 Join Algorithms

8. 5 points: For the query:

```
SELECT *
FROM items, suppliers, orders
WHERE items.iid = suppliers.iid AND suppliers.iid = orders.iid
AND suppliers.s_id = 1024;
```

The query optimizer produces the following query plan:


Grace Hash Join$_{orders.iid=suppliers.iid}$
Nested Loop Join$_{items.iid=suppliers.iid}$
Seq scan on orders
Seq scan on items
Index scan on suppliers s_id = '1024'

(a) Why are we using a nested loop join in items ⋈ suppliers?

The index lookup on suppliers will return one key (if it is a primary key) or very few keys otherwise. The nested loop join is cheaper here because its cost of comparison is less than that of the hash join.

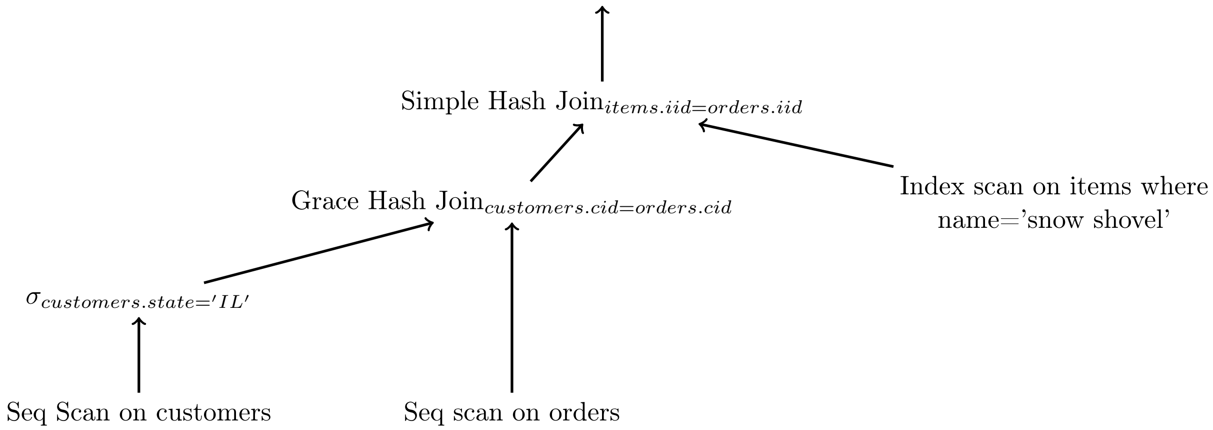
(b) Explain the use of a grace hash join in the second join.

Orders and suppliers have a N:N relationship and will have a large number of comparisons. We do this in lieu of the sort-merge join to save on sorting time. It is also faster than nested loop join because there are fewer comparisons.

9. 5 points: For the query:

```
SELECT *
FROM customers, orders, items
WHERE customers.cid = orders.cid AND items.iid = orders.iid
AND customers.state = 'IL' and items.name = 'snow shovel';
```

The query optimizer produces the following query plan:


Simple Hash Join$_{items.iid=orders.iid}$
Grace Hash Join$_{customers.cid=orders.cid}$
Index scan on items where name='snow shovel'
$\sigma_{customers.state='IL'}$
Seq Scan on customers
Seq scan on orders

(a) Why are we using a grace hash join in customers ⋈ orders?

The logic is the same as above.

(b) Justify the use of a simple hash join in the second join.

We use a simple hash join for the second one because the filter for items named 'snow shovel' is likely to result in few partitions for the hash join. Therefore, we keep this intermediate result in memory, saving on I/O time.

V Indexing and Access Paths

10. 5 points: We are querying the employee table of a business with the following schema:

`employee (empl_id, name, salary, dept_no, building_no, managed_by)`

For the following scenarios, which index will be the fastest?

1. B+ tree
2. Extendible hash
3. Bitmap index
4. Sequential Scan

The company has three buildings, and each has an equal number of employees. The table is much bigger than memory and clustered on its primary key, employee id. It has 1000 managers, each having a different number of direct reports in `managed_by`.

Match each of the letters above with one scenario below:

(a) Count the number of employees in the Glass building. **3**

(b) Sum up the total salaries paid over all employees. **4**

(c) Look up specific employees one at a time by id to give them raises. **1**

(d) List the employees managed by John. **2**

VI Query Planning

Consider the following query:

```
SELECT *
FROM A, B, C
WHERE A.v = B.v AND A.w = C.w
AND A.x > 7 and B.v = 5
AND C.w <= 3;
```

Assume that all of the attributes have a range of 1...10, inclusive. A has 100 tuples, B has 200, and C has 500.

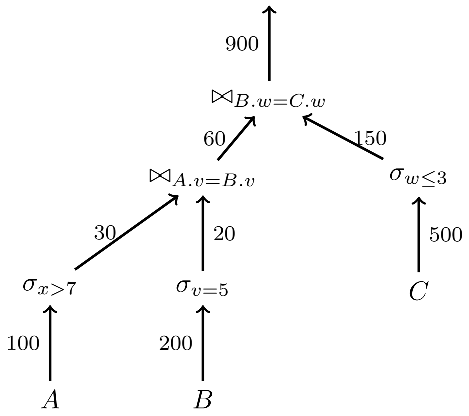
11. 5 points: Estimate the number of tuples that would be initially selected from each of the three relations if all of the non-join predicates are applied to them before any join processing begins.

A: $\frac{3}{10} \times 100 = 30$
B: $\frac{1}{10} \times 200 = 20$
C: $\frac{3}{10} \times 500 = 150$

12. 5 points: If we join in the order $A \bowtie B \bowtie C$, what is the expected output cardinality? Show your arithmetic.

$A \bowtie B : (30 \times 20) \times \frac{1}{max(10,1)} = 60$
$A \bowtie B \bowtie C : (60 \times 150) \times \frac{1}{max(10,3)} = 900$ (w is unfiltered for A)

13. 5 points: Draw a query plan tree for $A \bowtie B \bowtie C$. Label the estimated size of each intermediate result and the output cardinality.


900
$\bowtie_{B.w=C.w}$
60
150
$\bowtie_{A.v=B.v}$
$\sigma_{w\leq 3}$
30
20
500
$\sigma_{x>7}$
$\sigma_{v=5}$
C
100
200
A
B