EECS-6421: Exam

Electrical Engineering & Computer Science

Lassonde School of Engineering

York University

Family Name: _____

Given Name: _____

Student#: _____

EECS Account: _____

Instructor: Parke Godfrey
Exam Duration: take-home
Term: Fall 2017
Instructions

- **rules**
  - The test is open-note, open-book.
  - Use of a calculator is permitted.
- **answers**
  - Should you feel a question needs an assumption to be able to answer it, write the assumptions you need along with your answer.
  - This is an exam. Do not do work with others on it.
  - If you need more room to write an answer, indicate where you are continuing the answer. Add extra "blank" pages for space, if really needed
- **points**
  - The number of points a given question is worth is marked.
  - There are three major parts, 30 points in total.

Marking Box	
1.	/ 5
2.	/ 5
3.	/ 5
4.	/ 5
5.	/ 5
Total	/25

1. [5pt] **Datalog¬.** *What's that again, in* English *this time!?* [Exercise]

Consider (again) the schema

Movie(<u>title</u>, director, year)
Cast(<u>actor</u>, <u>title</u>, <u>role</u>)
FK (title) refs **Movie**

Consider the following rules which are used in the queries to follow.

castIn $(A,\ M) \leftarrow$ *cast* $(A,\ M,\ R)$.
actor $(A) \leftarrow$ *castIn* $(A,\ M)$.
castOut $(A,\ M) \leftarrow$ *castIn* $(A,\ M)$, *castIn* $(A,\ N)$, $M \neq N$.
diCast $(A,\ M) \leftarrow$ *cast* $(A,\ M,\ R_1)$, *cast* $(A,\ M,\ R_2)$, $R_1 \neq R_2$.

For each of the following Datalog¬ queries, restate the query in concise, understandable English.

(a) [1pt]

temp $(A) \leftarrow$ *diCast* $(A,\ M)$.
$\leftarrow$ *temp* (A).

(b) [1pt]

temp $(A) \leftarrow$ *actor* (A), *movie* $(M,\ D,\ Y)$, **not** *castIn* $(A,\ M)$.
$\leftarrow$ *actor* (A), **not** *temp* (A).

(c) [1pt]

$temp\ (A) \leftarrow castIn\ (A,\ M_1),\ castOut\ (A,\ M_2).$
$\leftarrow temp\ (A).$

(d) [1pt]

$temp\ (A) \leftarrow castIn\ (A,\ M),$ **not** $castOut\ (A,\ M).$
$\leftarrow temp\ (A).$

(e) [1pt]

$temp\ (A) \leftarrow castIn\ (A,\ M),$ **not** $diCast\ (A,\ M),$ **not** $castOut\ (A,\ M).$
$\leftarrow temp\ (A).$

2. [5pt] **Negation Semantics.** *A somewhat stable model.* [Concept]

(a) [3pt] Is there a Datalog¬ database **P** such that p (a *positive* atomic consequence) is a consequence of **P** with respect to the stable model semantics, but p is *not* a consequence of **P** with respect to the well-founded semantics, *and* **P** has a unique stable model?

If this cannot happen, explain why not. Otherwise, provide an example.

(b) [2pt] Is there a Datalog¬ database **P** such that p (a *positive* atomic consequence) is a consequence of **P** with respect to the well founded semantics, but p is *not* a consequence of **P** with respect to the stable model semantics? (Note that when **P** has *no* stable models, *everything* is a consequence of **P** with respect to the stable model semantics.)

If this cannot happen, explain why not. Otherwise, provide an example.

3. [5pt] **Sequential Reads.** *Speed it up.* [Analysis]

Consider each of the join algorithms that we have studied, in turn. Explain briefly whether sequential reads and writes would be advantageous in each case.

Assume that sequential reads are generally not advantageous for filescans of base tables. Base tables become fragmented on disk over time due to inserts and deletes.

(a) [1pt] **BNLJ** (block nested loop join)

(b) [1pt] **INLJ** (index-nested loop join)

(c) [1pt] **HJ** (two-pass hash join)

(d) [1pt] **SMJ** (two-pass sort merge join)

(e) [1pt] **MJ** (merge join, with outer and inner sorted prior)

4. [5pt] **Query Optimization.** *Simply the best plan available.* [Exercise]

Schema:

Employee(<u>eid</u>, name, did, jobCat, salary)
JobBenefits(<u>title</u>, <u>jobCat</u>, since)
FK (title) refs **Benefit**
Benefit(<u>title</u>, description, cost)

Statistics:

- **Employee**: 100,000 records on 2,000 pages
  - jobCat: 500 distinct values
  - did (department ID): 100 distinct values (department #13 is accounting)
- **JobBenefits**: 3,500 records on 70 pages
  - title: 200 distinct values
  - jobCat: 500 distinct values (same values as in **Employee**.jobCat)
- **Benefit**: 200 records on 20 pages
  - cost: ranges over $\$500, \ldots, \$10,500$

Indexes:

- **Employee**:
  - Clustered tree index on eid. (Index pages two deep; third layer, data-entry pages.)
  - Unclustered tree index on did, jobCat. (Index pages two deep; third layer, data-entry pages.)
- **JobBenefits**:
  - Clustered tree index on jobCat, title. (Index page one deep; second layer, data-entry pages.)
- **Benefit**:
  - Hash index on title.

Query:

```
select name, eid, B.title
    from Employee E, JobBenefits J, Benefit B
    where E.jobCat = J.jobCat
      and J.title = B.title
      and E.did = 13
      and B.cost > 10000;
```

You have an allocation of twelve buffer frames.

(a) [1pt] Estimate the cardinality—the resulting number of records—of the query.

(b) [4pt] Devise a good query plan for the query. Show the query tree, *fully* annotated with the chosen algorithms and access paths.

Estimate the cost of your plan. For full credit, you should have a plan that costs less than 1,500 I/O's.

5. [5pt] **MapReduce.** *Isn't this cartography?!* [Design]

Consider as input files that look as follows.

List

parke	ice cream
parke	tofu
parke	tomatoes
jeff	broccoli
jeff	hummus
jeff	ice cream
eric	milk
	⋮

Price

broccoli	3.99
hummus	4.50
ice cream	6.25
milk	4.99
tofu	3.59
tomatoes	5.00
⋮	

Input **List** represents grocery lists for people. Input **Price** yields the price for each grocery item. Assume the format of each is *key-value*.

(a) [2pt] Design a *MapReduce* job that outputs grocery *item* (e.g., "ice cream") as *key* and the *count* of the number of *people* as *value* (e.g., 2) who have that item on their list.

Make clear your *map* and *reduce* procedure pairs:

- the input of each *map*;
- the output key-value of each *map*;
- the output key-value of each *reduce*; *and*
- simple pseudo-code / clear description of what each *map* and *reduce* does.

(b) [2pt] Design a *MapReduce* job that outputs the total cost—sum of prices—for each person's grocery list. E.g.,

parke	14.84
jeff	14.74
eric	4.99
⋮	

Do *not* assume that the *values* are sorted by the shuffle; unlike in Project #2, your platform here *cannot* provide this. Note that a *map* can take its input from *more* than directory. Describe your *MapReduce* job as in Question 5a. You may assume that you have the output from the job in Question 5a in a directory *Count*. Be concerned that nearly everyone may have, say, *ice cream* on their list. (*Hint:* Think about *ice cream,1*, *ice cream,2*, ..., *ice cream,N* as keys.)

(c) [1pt] Can *map* and *reduce* filter keys, to eliminate keys that do not meet some condition?

Briefly explain why or why not.

Extra Space

Relax. Send in your test. Return to the wild!