CSE 344 Midterm

Wednesday, Nov. 1st, 2017, 1:30-2:20

Name: _____

Question	Points	Score
1	36	
2	29	
3	15	
4	20	
Total:	100	

- This exam is CLOSED book and CLOSED devices.
- You are allowed ONE letter-size page with notes (both sides).
- You have 50 minutes;
- Answer the easy questions before you spend too much time on the more difficult ones.
- Good luck!

1 SQL

1. (36 points)

   A company maintains a database about their employees and projects with the following schema.

   ```
   Employee(eid, name, salary)
   Project(pid, title, budget)
   WorksOn(eid, pid, year)
   ```

   `WorksOn` records which empoloyee worked on which projects. `salary` and `budget` represent yearly salary and budget respectively. An employee may work on multiple project during the same year, and may also work on the same project during multiple years. All keys are underlined, and `WorksOn.eid`, `WorksOn.pid` are foreign keys to `Employee` and `Project` respectively.

   (a) (10 points) The yearly salary expenses of a project is the sum of all salaries of the employees who worked on that project during that year. Write a SQL query to find all the projects whose yearly salary expenses exceeded its budget. Return only the project title, and sort the projects in ascending order alphabetically by their title.

   Solution:

   ```
   select distinct z.title
   from Employee x, WorksOn y, Project z
   where x.eid = y.eid and y.pid = z.pid
   group by z.pid, z.title, y.year, z.budget
   having sum(salary) > z.budget
   order by z.title;
   ```

   −1 for every missing GROUP BY attribute (max -2; grouping by pid optional)
   −1 missing SELECT DISTINCT
   −2 unnecessary subquery (-1 for subquery that returns correct results/compiles)
   −1 missing ORDER BY
   −1 missing aggregates
   −2 missing tables
   −2 missing joins
   −1 using WHERE instead of HAVING
   −1 selecting incorrect attributes
   −1 for various other errors

```
Employee(eid, name, salary)
Project(pid, title, budget)
WorksOn(eid, pid, year)
```

(b) (10 points) We say that an employee worked intermittently on a project p if she worked on p during one year, then did not work on p during a later year, then worked again on p during an even later year. For example if Alice worked on the project during 2012, 2013, and 2017 then we say that she worked intermittently on p; if Bob worked on that project during 2013, 2014, 2015 and no other years, then we say he worked continuously. Write a SQL query to retrieve all employees that worked intermittently on some project. Return the employee name, and the project title.

Solution:

```
select distinct u.name, v.title
from Employee u, WorksOn x, WorksOn y, Project v
where u.eid = x.eid and u.eid = y.eid
  and x.pid = v.pid and y.pid = v.pid
  and x.year + 1 < y.year
  and not exists (select *
                  from WorksOn z
                  where u.eid = z.eid and v.pid = z.pid
                    and x.year < z.year and z.year < y.year);
```

(1 point) Distinct output generated
(3 Points) Basic query structure
- Selected the name and title
- FROM included all tables
- Proper joins created between tables

(1 point) Included check for year separation (ex: $year1 > year2 + 1$)
(5 points) Verification of intermittency (ex: subquery to check for years between separated years)

There were also variations on the solutions that could have worked (or did work). We took points off as similar to the original rubric as we could.

```
Employee(eid, name, salary)
Project(pid, title, budget)
WorksOn(eid, pid, year)
```

(c) For each question below indicate whether the two SQL queries are equivalent. Assume that the database does not contain any NULL values.

i. (2 points) Are Q1 and Q2 equivalent?

```
Q1: select W.year, W.pid, count(*)
    from WorksOn W
    group by W.year, W.pid;

Q2: select distinct W.year, W.pid,
                   (select count(*)
                    from WorksOn W2
                    where W.year=W2.year and W.pid = W2.pid)
    from WorksOn W;
```

i. **Yes**

Yes/No:

ii. (2 points) Are Q3 and Q4 equivalent?

```
Q3: select W.year, W.pid, count(*)
    from WorksOn W, Employee E
    where W.eid = E.eid and E.salary > 100000
    group by W.year, W.pid;

Q4: select W.year, W.pid,
           (select count(*)
            from Employee E
            where W.eid = E.eid and E.salary > 100000)
    from WorksOn W;
```

ii. **No**

Yes/No:

iii. (2 points) Are Q5 and Q6 equivalent?

```
Q5: select distinct E.name
    from Employee E, WorksOn W, WorksOn W2
    where E.eid = W.eid and E.eid = W2.eid
      and W.pid = W2.pid
      and W.year > 2010 and W2.year < 2015;

Q6: select distinct E.name
    from Employee E, WorksOn W
    where E.eid = W.eid
      and W.year > 2010;
```

iii. **No**

Yes/No:

iv. (2 points) Are Q7 and Q8 equivalent?

```
Q7: select distinct E.name
    from Employee E, WorksOn W, WorksOn W2
    where E.eid = W.eid and E.eid = W2.eid
      and W.pid = W2.pid
      and W.year < 2010 and W2.year < 2015;

Q8: select distinct E.name
    from Employee E, WorksOn W
    where E.eid = W.eid
      and W.year < 2010;
```

iv. **Yes**

Yes/No:

```
Employee(eid, name, salary)
Project(pid, title, budget)
WorksOn(eid, pid, year)
```

(d) Consider the following database instance:

Employee

eid	name	salary
1	Alice	1000
2	Bob	NULL

WorksOn

eid	pid	year
1	10	NULL
1	20	2015
2	20	NULL

Project

pid	title	budget
10	OS	NULL
20	ML	5000

Solution:

```
delete from WorksOn;
delete from Employee;
delete from Project;
insert into Employee values(1, 'Alice', 1000);
insert into Employee values(2, 'Bob', NULL);
insert into Project values(10, 'OS', NULL);
insert into Project values(20, 'ML', 5000);
insert into WorksOn values (1,10,NULL);
insert into WorksOn values (1,20,2015);
insert into WorksOn values (2,20,NULL);
```

Indicate for each query below what answers it returns; write "empty" if the answer is the emptyset.

i. (2 points) What does query Q1 return?:

```
Q1: select x.name, z.title
    from Employee x, WorksOn y, Project z
    where x.eid = y.eid and y.pid = z.pid
      and (y.year = 2015 or y.year != 2015);
```

Solution:

Alice	ML

ii. (2 points) What does query Q2 return?:

```
Q2: select x.name, z.title
    from Employee x, WorksOn y, Project z
    where x.eid = y.eid and y.pid = z.pid
      and ( (salary = 1000 and year = 2015) or budget = 5000);
```

Solution:

Alice	ML
Bob	ML

Employee

eid	name	salary
1	Alice	1000
2	Bob	NULL

WorksOn

eid	pid	year
1	10	NULL
1	20	2015
2	20	NULL

Project

pid	title	budget
10	OS	NULL
20	ML	5000

iii. (2 points) What does query Q3 return?:

```
Q3: select x.name, z.title
    from Employee x, WorksOn y, Project z
    where x.eid = y.eid and y.pid = z.pid
      and (salary = 1000 or budget = 5000);
```

Solution:

Alice	OS
Alice	ML
Bob	ML

iv. (2 points) What does query Q4 return?:

```
Q4: select x.name, z.title
    from Employee x, WorksOn y, Project z
    where x.eid = y.eid and y.pid = z.pid
      and ((salary = 1000 or year = 2015) and not(budget = 5000));
```

Solution: Empty

2 Relational Algebra

2. (29 points)

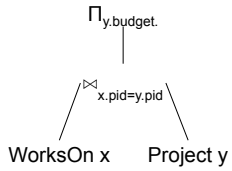
Consider the same relational schema as before, including the key/foreign key constraints:

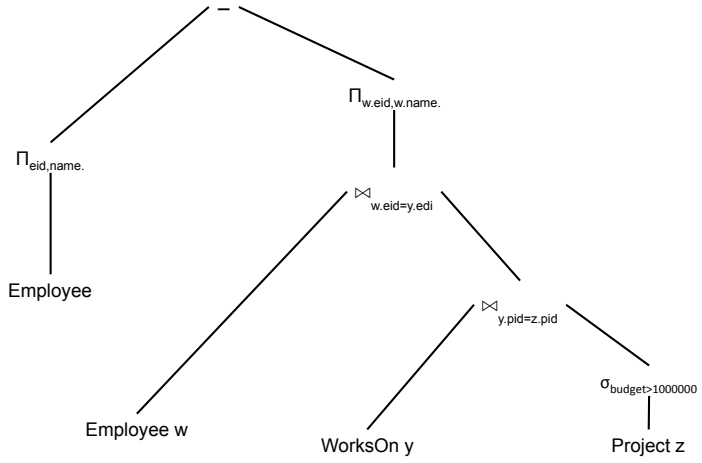
```
Employee(eid, name, salary)
Project(pid, title, budget)
WorksOn(eid, pid, year)
```

(a) (5 points) Write a Relational Algebra expression in the form of a logical query plan (i.e., draw a tree) that is equivalent to the SQL query below. Your query plan does not have to be necessarily "optimal": however, points will be taken off for overly complex solutions.

```
select x.eid, x.name
from Employee x
where not exists (select *
                  from WorksOn y, Project z
                  where x.eid = y.eid and y.pid = z.pid
                   and z.budget > 1000000);
```

Hint: to avoid renaming, use aliases in the query plan, like this





Solution: −1: Missing or incorrect projection

−1: Did not join another Employee table to recover the name information

−0.5: Overly complicated query but correct
−1: Missing or incorrect usage of set/bag difference
−0.5: Small mistakes on the operator (Wrong ranges, subscript etc.)

(b) Assume that the database instance does not contain NULL's, but otherwise can be any valid instance (satisfying all key/foreign key constraints). Answer the questions below, when the relational algebra expressions have bag semantics.

i. (2 points) Are these two expressions equivalent?

$$\Pi_{\texttt{year}}(\sigma_{\texttt{salary}>40000}(\texttt{Employee} \bowtie_{\texttt{eid=eid}} \texttt{WorksOn})) = \Pi_{\texttt{year}}(\texttt{WorksOn})$$

i. **No**

Yes/No:

ii. (2 points) Are these two expressions equivalent?

$$\Pi_{\texttt{year}}(\sigma_{\texttt{year}>2015}(\texttt{Employee} \bowtie_{\texttt{eid=eid}} \texttt{WorksOn})) = \Pi_{\texttt{year}}(\sigma_{\texttt{year}>2015}(\texttt{WorksOn}))$$

ii. **Yes (no NULL)**

Yes/No:

iii. (2 points) Are these two expressions equivalent?

$$\Pi_{\texttt{name}}(\sigma_{\texttt{salary}>40000}(\texttt{Employee} \bowtie_{\texttt{eid=eid}} \texttt{WorksOn})) = \Pi_{\texttt{name}}(\sigma_{\texttt{salary}>40000}(\texttt{Employee}))$$

iii. **No (bag semantics)**

Yes/No:

iv. (2 points) Are these two expressions equivalent?

$$\Pi_{\texttt{name}}(\sigma_{\texttt{year}>2015}(\texttt{Employee} \bowtie_{\texttt{eid=eid}} \texttt{WorksOn})) = \Pi_{\texttt{name}}(\texttt{Employee})$$

iv. **No**

Yes/No:

v. (2 points) The notation $|S|$ means the cardinality of a set S (number of tuples in S). Does the following always hold?

$$|\texttt{Employee} \bowtie_{\texttt{eid=eid}} \sigma_{\texttt{year}<2015}(\texttt{WorksOn})| \leq |\texttt{Employee}|$$

v. **No**

Yes/No:

vi. (2 points) Does the following always hold?

$$|\sigma_{\texttt{salary}<5000}(\texttt{Employee}) \bowtie_{\texttt{eid=eid}} \texttt{WorksOn}| \leq |\texttt{WorksOn}|$$

vi. **Yes**

Yes/No:

(c) For each statement below, indicate whether it is true or false.

i. (2 points) If the attribute K of a relation is a key, then no two tuples in the relation can have the same value of K.

True/False:

i. **true**

ii. (2 points) If the attribute K of a relation is a foreign key, then no two tuples in the relation can have the same value of K.

True/False:

ii. **false**

iii. (2 points) *First normal form* means that all relations in the database are flat.

True/False:

iii. **true**

iv. (2 points) *Physical data independence* means that the relations in database are physically independent of each other.

True/False:

iv. **false**

v. (2 points) All queries expressible in Relational Algebra are monotone.

True/False:

v. **false**

vi. (2 points) All queries expressible in datalog (with recursion, but without negation and without aggregates) are monotone.

True/False:

vi. **true**

3 Datalog

```
Employee(eid, name, salary)
Project(pid, title, budget)
WorksOn(eid, pid, year)
```

3. (15 points)

   Answer the questions below.

   (a) (5 points) Write a datalog program that returns all employees that worked only on the ‘‘Compiler’’ project. Your query should return the `eid` and `name` of each such employee.

Solution:

```
nonAnswer(eid) :- Employee(eid, -, -),
                  WorksOn(eid, pid, -),
                  Project(eid, t, -),
                  t!= 'Compiler'
Answer(eid, name) :- Employee(eid, name, -), !nonAnswer(eid)
```

2-3 points off for missing negation
1 point off for failing to return employees without any projects
2 points off for unsafe query

```
Employee(eid, name, salary)
Project(pid, title, budget)
WorksOn(eid, pid, year)
```

(b) (10 points) A project p1 influences a project p2, if there exists an employee who worked on p1 during some year, then worked on p2 during some later year. After a major bug was discovered in several projects, the company traced it down to a design flaw in the ``Compiler'' project, and now wants to retain only the projects that were not influenced by ``Compiler''. (Note ``Compiler'' *is* influenced by ``Compiler''.) Write a datalog program to find all projects who were not influenced by the ``Compiler'' project; return their `pid` and `title`.

Solution:

```
Infl(x) :- Project(x, 'Compiler', -)
Infl(z) :- Infl(x),
           WorksOn(eid, x, y1),
           WorksOn(eid, z, y2),
           y1 < y2
Answ(pid, title) :- Project(pid, title, -), !Infl(pid)
```

5 points off for missing recursion (almost nobody wrote a recursive query!)
2 points off for unsaafe
1 point off for missing $y1 < y2$
1 point off for missing negation

4 Semistructured Data

4. (20 points)

(a) Answer the following questions. In your answer you may omit apostrophes, i.e. you may write {A:a1} instead of {'A':'a1'} .

i. (2 points) What does the following SQL++ query return?

```
with t as
 [{'A':'a1', 'F':[{'B':'1'}, {'B':'2'}], 'G':[{'C':'1'},{'C':'2'}]},
  {'A':'a2', 'F':[{'B':'3'}, {'B':'4'},{'B':'5'}], 'G':[ ]},
  {'A':'a3', 'F':[{'B':'2'}], 'G':[{'C':'1'},{'C':'3'}]}]
Select x.A
From  t  x, x.G y
where y.C='1';
```

Solution:

```
{ "A": "a1" }
{ "A": "a3" }
```

ii. (2 points) What does the following SQL++ query return?

```
with t as
 [{'A':'a1', 'F':[{'B':'1'}, {'B':'2'}], 'G':[{'C':'1'},{'C':'2'}]},
  {'A':'a2', 'F':[{'B':'3'}, {'B':'4'},{'B':'5'}], 'G':[ ]},
  {'A':'a3', 'F':[{'B':'2'}], 'G':[{'C':'1'},{'C':'3'}]}]
Select x.A, y.B
From  t  x, x.F y;
```

Solution:

```
{ "A": "a1", "B": "1" }
{ "A": "a1", "B": "2" }
{ "A": "a2", "B": "3" }
{ "A": "a2", "B": "4" }
{ "A": "a2", "B": "5" }
{ "A": "a3", "B": "2" }
```

iii. (2 points) What does the following SQL++ query return?

```
with t as
 [{'A':'a1', 'F':[{'B':'1'}, {'B':'2'}], 'G':[{'C':'1'},{'C':'2'}]},
  {'A':'a2', 'F':[{'B':'3'}, {'B':'4'},{'B':'5'}], 'G':[ ]},
  {'A':'a3', 'F':[{'B':'2'}], 'G':[{'C':'1'},{'C':'3'}]}]
Select x.A, y.B
From  t  x, x.F y, x.G z
where y.B=z.C ;
```

Solution:

```
{ "A": "a1", "B": "1" }
{ "A": "a1", "B": "2" }
```

iv. (2 points) What does the following SQL++ query return?

```
with t as
 [{'A':'a1', 'F':[{'B':'1'}, {'B':'2'}], 'G':[{'C':'1'},{'C':'2'}]},
  {'A':'a2', 'F':[{'B':'3'}, {'B':'4'},{'B':'5'}], 'G':[ ]},
  {'A':'a3', 'F':[{'B':'2'}], 'G':[{'C':'1'},{'C':'3'}]}]
Select x1.A as A1, x2.A as A2, y.B
From  t x1, t x2, x1.F y, x2.G z
where y.B=z.C ;
```

Solution:

```
{ "A1": "a1", "A2": "a1", "B": "1" }
{ "A1": "a1", "A2": "a3", "B": "1" }
{ "A1": "a1", "A2": "a1", "B": "2" }
{ "A1": "a2", "A2": "a3", "B": "3" }
{ "A1": "a3", "A2": "a1", "B": "2" }
```

(b) For each statement below, indicate whether it is true or false.

i. (2 points) An *OLTP workload* means a workload of queries that have many joins, aggregates, and very few or no updates.

i. **false**

True/False:

ii. (2 points) An *OLAP workload* means a workload of queries that have many joins, aggregates, and very few or no updates.

ii. **true**

True/False:

iii. (2 points) NoSQL systems are primarily intended for OLTP workloads, and not for OLAP workloads.

iii. **true**

True/False:

iv. (2 points) NoSQL systems emerged because SQL is a very old language and needs to be replaced by something more modern.

iv. **false**

True/False:

v. (2 points) NoSQL systems typically run on a large, distributed cluster (meaning: many servers).

v. **true**

True/False:

vi. (2 points) NoSQL systems support physical data independence better than relational database systems.

vi. **false**

True/False: