## 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;
```

Yes/No:

iii. **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;
```

Yes/No:

iv. **Yes**

## 1 SQL and Indexing

1. (50 points)

   Consider the following database about picture tagging Website:

   Member(<u>mid</u>, name, age)
   Picture(<u>pid</u>, year)
   Tagged(mid, pid)

**Solution:**

```
drop table if exists Tagged;
drop table if exists Member;
drop table if exists Picture;

create table Member(mid int, name text, age int);
create table Picture(pid int, year int);
create table Tagged(mid int, pid int);

insert into Member values(1, 'Alice', 10);
insert into Member values(2, 'Bob', 20);
insert into Member values(3, 'Carol', 30);

insert into Picture values(10, 2011);
insert into Picture values(20, 2012);
insert into Picture values(30, 2013);
insert into Picture values(40, 2014);

insert into Tagged values(1,10);
insert into Tagged values(1,20);
insert into Tagged values(2,20);
insert into Tagged values(3,10);
insert into Tagged values(3,20);
insert into Tagged values(3,30);
```

Relation Member contains a tuple for each registered member of the Website; Picture contains the metadata of the pictures (its key pid, and the year when the picture was taken), and Tagged says which user was tagged in what picture. Primary keys are underlined. Attributes values may be null.

(a) (10 points) Write a SQL query that retrieves the names of all members that were tagged both in the year 2011 and in 2014. Return them sorted in lexicographic order.

**Solution:**

```
select x.name
from Member x, Tagged y1, Picture z1, Tagged y2, Picture z2
```

```
where x.mid = y1.mid and y1.pid = z1.pid and z1.year = 2011
  and x.mid = y2.mid and y2.pid = z2.pid and z2.year = 2014
order by x.name;
```

Common mistakes:

1. Joining both Picture Tables to the same Tagged Table.

```
select M.name
from Member M, Tagged T, Picture P1, Picture P2
where M.mid = T.mid and P1.pid = T.pid and P2.pid = T.pid and
P1.year = 2011 and P2.year = 2014
order by M.name;
```

`T.pid = P1.pid and T.pid = P2.pid => P1.pid = P2.pid => P1.year = P2.year`
since pid is the primary key of Picture

Students received 5 points for this answer.

2. Selecting Members who were tagged in 2011 or 2014 instead of both.

```
select M.name
from Member M, Tagged T, Picture P,
where M.mid = T.mid and P.pid = T.pid and
(P.year = 2011 or P.year = 2014)
order by M.name;
```

Students received 3 points for this answer.

```
Member(mid, name, age)
Picture(pid, year)
Tagged(mid, pid)
```

(b) (10 points) For each query below, indicate if the query correctly returns the `name` of all users who were never tagged in 2015. Anywhere between zero and all queries compute the correct answer. Each query is syntactically correct.

```
Q1 = select distinct x.name
     from Member x, Tagged y
     where x.mid = y.mid
       and not exists
                 (select *
                  from Picture z
                  where y.pid = z.pid
                    and z.year = 2015);

Q2 = select distinct x.name
     from Member x
     where not exists
                (select *
                 from Tagged y, Picture z
                 where x.mid = y.mid
                   and y.pid = z.pid
                   and z.year = 2015);

Q3 = select distinct x.name
     from Member x
     where not exists
            (select *
             from Tagged y
             where x.mid = y.mid
              and not exists
                  (select *
                   from Picture z
                   where y.pid = z.pid
                     and z.year = 2015));

Q4 = select distinct x.name
     from Member x, Tagged y, Picture z
     where x.mid = y.mid and y.pid = z.pid and z.year = 2015
     group by x.name
     having count(z.pid) = 0;
```

(b) **Q2**

Answer with any subset of Q1, Q2, Q3, Q4:

```
Member(mid, name, age)
Picture(pid, year)
Tagged(mid, pid)
```

(d) (10 points) Write a SQL query that returns the mid's and names of all members that were tagged only in pictures were Alice was also tagged. Hint: you may want to first answer question 2.b.

**Solution:** Datalog:

```
aliceTagged(pid) :- Member(mid, 'Alice',-),Tagged(mid, pid)
nonAnswer(mid) :- Tagged(mid,pid) not aliceTagged(pid)
answer(mid,name) :- Member(mid,name,-), not nonAnswer(mid)
```

```
select x.mid, x.name
from Member x
where x.mid not in
           (select y.mid
            from Tagged y
            where y.pid not in
               (select v.pid from Member u,Tagged v
                where u.name='Alice' and u.mid=v.mid));
```

Common mistakes:

1. Have a single level of negation.
2. Have a single level of negation combined with `u.name!='Alice'`

Students received 4 points for such solutions.

# 2 Relational Algebra

2. (29 points)

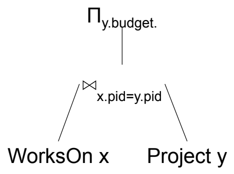
Consider the same relational schema as before, including the key/foreign key constraints:

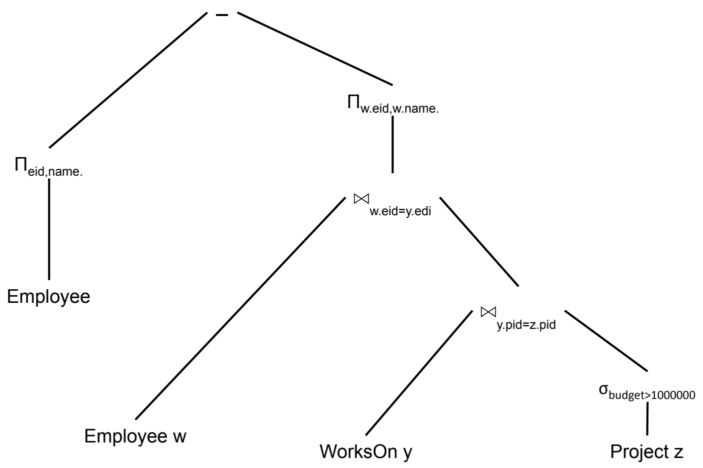
```
Employee(eid, name, salary)
Project(pid, title, budget)
WorksOn(eid, pid, year)
```

(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?

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

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?

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

v. **No**

Yes/No:

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

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

vi. **Yes**

Yes/No:

## Problem 2: Writing Queries (52 points total)

Write the following queries using the schema below:

```
Product (pid, name, cid)  -- cid is foreign key to Company.cid
Company (cid, cname, city)
Purchase(pid, custId, quantity, price)
-- pid is foreign key to Product.pid, custId is foreign key to Customer.custId
Customer(custId, name, city)
```

a) Write a SQL query that returns the name of companies, along with the number of products sold, for companies that have sold at least 2 different types of products anywhere. (13 points)

```
SELECT c.cname, count(*)
FROM product p, company c, purchase p2
WHERE p.pid = p2.pid and c.cid = p.cid
GROUP BY c.cid, c.name
HAVING COUNT(*) > 2
```

b) Write a relational algebra query that returns the distinct names of all customers from Seattle who purchased any one type of product with quantity > 10. Write your query as a tree or a single relational algebra expression. (13 points)

$\delta(\pi_{name}(\sigma_{city="Seattle"}(\text{customer}) \bowtie_{custId=custId} \sigma_{quantity>10}(\text{purchase})))$

## 3 Relational Data Model and Query Evaluation

3. (30 points)

   Answer each question below.

   (a) (2 points) In the relational data model an attribute of a relation cannot be another relation.

   (a) **True**

   True or false?

   (b) (2 points) Consider the relation `Product(productName, manufacturer, price, weight)` where the key is `(productName, manufacturer)`. Can there be two different products with the same value of `manufacturer`?

   (b) **Yes**

   Yes or no?

   (c) (2 points) If an attribute in a table is a foreign key, then the table cannot contain two tuples with the same value of that attribute.

   (c) **False**

   True or false?

   (d) (2 points) In this and the following question we will assume that the relational algebra has set semantics. If the relation $R(A, B)$ has $m$ tuples, and the relation $S(B, C)$ has $n$ tuples, what is the largest possible size of the output of the natural join $R \bowtie S$? Choose one of the following:

   (a) $m$
   (b) $n$
   (c) $mn$
   (d) $m + n$
   (e) $\max(m, n)$
   (f) $(m + n)/2$
   (g) None of the above.

   (d) **c**

   Answer a-g:

   (e) (2 points) If the relation $R(A, B)$ has $m$ tuples, and the relation $S(B, C)$ has $n$ tuples, what is the largest possible size of the output of $\Pi_{AB}(R \bowtie S)$? Here $\bowtie$ represents the natural join. Choose from the same options as in the previous question.

   (e) **c**

   Answer a-g:

(f) Consider two relations $R(A, B), S(C, D)$, where all attributes are integers and cannot be NULL. For each identity below, indicate whether it is true or false.

i. (2 points) $R \bowtie_{B=C} S = S \bowtie_{B=C} R$

i. **True**

True or false?

ii. (2 points) $(\sigma_{B<1000}(R)) \bowtie_{B\neq C} S = (\sigma_{B<1000}(R)) \bowtie_{B\neq C} (\sigma_{C\geq 1000}(S))$

ii. **False**

True or false?

iii. (2 points) $R \times S - R \bowtie_{B=C} S = R \bowtie_{B\neq C} S$

iii. **True**

True or false?

iv. (2 points) $R - \Pi_{AB}(R \bowtie_{B=C} S) = \Pi_{AB}(R \bowtie_{B\neq C} S)$

iv. **False**

True or false?

v. (2 points) $\gamma_{A,\text{count}(*)\to F}(R \bowtie_{B=C} S) = \gamma_{A,\text{sum(E)}\to F}(R \bowtie_{B=C} (\gamma_{C,\texttt{count(*)}\to E}(S)))$

v. **True**

True or false?

(g) (2 points) If a relation is stored in 1000 blocks on disc, then it cannot have more than 1000 records.

(g) **False**

True or false?

(h) (2 points) Alice is a database administrator. She wants to add indexes to a table that is accessed frequently by her applications. She knows that a clustered index delivers better performance than an unclustered index, but most indexes she creates are unclustered. Why? Choose one of the following:

(a) Because clustered indexes take more space than unclustered indexes and Alice's database servers has little space.

(b) Because a relation can have only one clustered index, but many unclustered indexes.

(c) Because clustered indexes will slow down updates and Alice's application has many updates.

(d) None of the above.

(h) **b**

Choose one of a-d:

(i) (2 points) There is an important distinction between logical operators and physical operators. Indicate which of the operators below are physical operators:

(a) Theta join
(b) Hash join
(c) Natural join
(d) Eq join
(e) Index-based join
(f) Merge join

(i) **b,e,f**

Answer a-f (choose all that apply):

(j) (2 points) Did the speed of sequential reads from disk increase significantly over the years?

(j) **Yes**

Yes or no?

(k) (2 points) Did the speed of random reads from disk increase significantly over the years?

(k) **No**

Yes or no?

(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:

## Problem 1: Warm up (10 points total)

Select either True or False for each of the following questions. For each question you get 1 point for answering it correctly, -0.5 point for an incorrect answer, and 0 point for no answer. The minimum you will get for this entire problem is 0.

a) The arity of the relation R(A int, B int) with tuples (4,2), (2,2), (4,2), (3,3), (4,2) is 3.

True [✓ **False**]

b) Data is encoded in JSon documents using key-value pairs.

[✓ **True**] False

c) A relation can have a clustered index on a set of attributes.

[✓ **True**] False

d) A datalog rule is safe if and only if every variable appears in a relational atom.

True [✓ **False**]

e) A SQL query with `GROUP BY` can select all attributes that are grouped on, joined upon, or are aggregates.

True [✓ **False**]

f) Every natural join can be rewritten using cross product and selection.

True [✓ **False**]

g) A foreign key does not have to reference a primary key.

[✓ **True**] False

h) An inner join between relations R and S always includes all tuples from R in the result.

True [✓ **False**]

i) Queries with universal quantifiers cannot be unnested.

[✓ **True**] False

j) A subquery in the `SELECT` clause in SQL can refer to tuple variables defined in the `WHERE` clause.

[✓ **True**] False