MASTER

EECS-3421N: TEST #2

Electrical Engineering & Computer Science

Lassonde School of Engineering

York University

Family Name: _____

Given Name: _____

Student#: _____

EECS Account: _____

Instructor: Parke Godfrey
Exam Duration: 75 minutes
Term: Winter 2019

Instructions

- **rules**
  - The test is closed-note, closed-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.
  - If you need more room to write an answer, indicate where you are continuing the answer.
  - For multiple choice questions, choose *one* best answer for each of the following. There is no negative penalty for a wrong answer.
- **notation**
  - For schema, the underlined attributes indicate a table's primary key (and are, hence, not nullable). Attributes in *italics* are not nullable. Foreign keys are indicated by FK.
  - Assume *set* semantics for relational-algebra expressions.
- **points**
  - The number of points a given question is worth is marked.
  - There are five major parts worth 10 points each, for 50 points in total.

MARKING BOX		
1.	CBBAB CEBED	/10
2.		/10
3.		/10
4.		/10
5.		/10
Total		/50

V3 Given

Errata

- Q4d (p. 9)
  P.prod# ⇒ A.prod#
- Q2d (p. 5)
  name ⇒ pname
- Q3c (p. 7)
  c.p# ⇒ c.actor
- Q1g B. (R ⋈ (T ⋈ R)) ⇒ (R ⋈ (T ⋈ S))

HAPPY PI DAY!

1. [10pt] **General.** *Much choice!* MULTIPLE CHOICE

Choose *one* best answer for each of the following. Each is worth one point. There is no negative penalty for a wrong answer.

In the *rare* case that you feel a clarification to your answer is needed, write a brief clarification on the side.

Let $|\mathbf{T}|$ denote the number of tuples in $\mathbf{T}$.

(a) [1pt] `SQL` is derived from

A. the *relational algebra.*
B. the *domain relational calculus.*
C. the *tuple relational calculus.*
D. XQuery for XML.
E. *`SQL` is a programming language, not a query language. It is not derived from any of the above.*

(b) [1pt] The *relational algebra* is an algebra because

A. it contains operators for arithmetic.
B. its operators map from the domain of operands back into the domain.
C. it allows one to order freely its operators to form expressions.
D. it contains a fixed, finite number of pre-defined operators.
E. *It is not really an algebra. But calling it an algebra sounds cool!*

(c) [1pt] In a relational database system, if you join (natural join) tables $\mathbf{R}$ and $\mathbf{S}$, but $\mathbf{R}$ is empty (that is, it has no tuples),

A. the system reports an error.
B. the answer set is an empty table.
C. the answer set is the same as table $\mathbf{S}$.
D. the answer set consists of just *one* row.
E. an answer set is returned; however, the results are system dependent.

(d) [1pt] $\mathbf{R} \cap \mathbf{S}$ is equivalent to

A. $\mathbf{R} - (\mathbf{R} - \mathbf{S})$
B. $\mathbf{R} - (\mathbf{S} - \mathbf{R})$
C. $(\mathbf{R} - \mathbf{S}) - \mathbf{R}$
D. $-((-\mathbf{R}) \cup (-\mathbf{S}))$
E. *There is not enough information to answer this.*

(e) What does the query "`select max(R.B) from R;`" return if $\mathbf{R}$ is empty?

A. An empty table of one column.
B. A table of one column with one row with the value ⟨NULL⟩.
C. A table of one column with one row with the value ⟨INF⟩.
D. An error message.
E. *Not enough information to determine.*

(f) [1pt] In *relational algebra*, we can write any expression using *join* ("⋈") using the following instead.

A. π, $\cap$
B. π, $\cup$
C. π, σ, $\times$
D. π, $-$
E. *Join cannot be rewritten with any combination of the other R.A. operators.*

(g) [1pt] Which of the following may evaluate to a different answer than $\mathsf{R} \bowtie (\mathsf{S} \bowtie \mathsf{T})$?

A. $(\mathsf{R} \bowtie \mathsf{S}) \bowtie \mathsf{T}$
B. $\mathsf{R} \bowtie (\mathsf{T} \bowtie \mathsf{R})$
C. $(\mathsf{S} \bowtie \mathsf{T}) \bowtie \mathsf{R}$
D. $\mathsf{T} \bowtie (\mathsf{S} \bowtie \mathsf{R})$
E. *They all evaluate the same.*

* Counted "B" as correct retroactively.

(h) [1pt] Consider the schema

$\mathsf{R}(\underline{\mathsf{A}}, B)$ FK (B) refs R (A)

What is the *largest* that $|\mathsf{R} \bowtie \pi_{\mathsf{A}\to\mathsf{B},\mathsf{B}\to\mathsf{A}}(\mathsf{R})|$ can be?

A. 0
B. $|\mathsf{R}|$
C. $\frac{1}{2}|\mathsf{R}|$
D. $2|\mathsf{R}|$
E. $|\mathsf{R}|^2$

(i) [1pt] Consider table $\mathsf{R}(\underline{\mathsf{A}}, \mathsf{B})$ for which B is of type *integer* and $|\mathsf{R}| = n > 0$. How many tuples will the query

```
select A from R where B <= 13 or B > 13;
```

return?

A. 0
B. $\frac{1}{2}n$
C. n
D. n^2
E. *There is not enough information to answer this.*

(j) [1pt] Consider the relations $\mathsf{R}(\underline{\mathsf{A}}, \underline{\mathsf{B}})$, $\mathsf{S}(\underline{\mathsf{B}}, \underline{\mathsf{C}})$, and $\mathsf{T}(\underline{\mathsf{C}}, \underline{\mathsf{A}})$.
One of these is not like the others. That is, one can evaluate differently than the other four. Which one?

A. $\pi_{\mathsf{A},\mathsf{B}}((\mathsf{R} \bowtie \mathsf{S}) \bowtie \mathsf{T})$
B. $\pi_{\mathsf{A},\mathsf{B}}(\mathsf{R} \bowtie (\mathsf{T} \bowtie \mathsf{S}))$
C. $\mathsf{R} \bowtie \pi_{\mathsf{A},\mathsf{B}}(\mathsf{S} \bowtie \mathsf{T})$
D. $\pi_{\mathsf{A},\mathsf{B}}(\mathsf{R} \bowtie \mathsf{T}) \bowtie \pi_{\mathsf{A},\mathsf{B}}(\mathsf{R} \bowtie \mathsf{S})$
E. $\pi_{\mathsf{A},\mathsf{B}}(\mathsf{R} \bowtie \mathsf{S}) \bowtie \pi_{\mathsf{A},\mathsf{B}}(\mathsf{S} \bowtie \mathsf{T})$

---

2. [10pt] **Relational Algebra.** *Riddle me this!* SHORT ANSWER

For Questions 2a to 2d, use the *Colours* schema in Figure 1 on page 13 (as used in examples in class) to write *relational-algebra* queries for the English questions posed and vice versa.

(a) [2pt] Show products by prod# and pname that come in colour *orange*.

$\Pi_{prod\#, pname}(Product \bowtie \sigma_{colour = 'orange'}(Avail\_Colours))$

+1: Correct join
+1: Correct Π & σ

(b) [2pt] Show products by prod# and pname that are owned by at least two customers.

$\Pi_{prod\#, pname}(Product \bowtie (\Pi_{cust\# \neq cust\#2, \overset{prod\#}{}}(Item \bowtie \Pi_{cust\# \to cust\#2, prod\#, pname}(Item))))$

+1: correct sources & Π's
+1: proper self-join on Item

(c) [2pt] Show pairs of customers—*first* (cust#) and *fname* (cname) for the first customer and *second* (cust#) and *sname* (cname) for the second customer—such that the second customer owns an item in the first customer's favourite colour.

$\Pi_{First, Fname, second, sname}(\sigma_{First <> second}($
$\Pi_{cust\# \to First, cname \to Fname, FavColour \to colour}(Customer)$
$\bowtie$
$\Pi_{cust\# \to second, cname \to sname, colour}($
$Item \bowtie Customer$ $)))$

+1: Two joins are correct, and on right attr's
+1: Projections correct

[2pt] Consider the following R.A. expression.

$$\pi_{\text{colour}}(\pi_{\text{colour,prod\#,name}}(\textbf{Product} \bowtie \textbf{Avail\_colour}) - \pi_{\text{colour,prod\#,name}}(\sigma_{\text{cost}>\text{cost2}}(\pi_{\text{colour,prod\#,name,cost}}(\textbf{Product} \bowtie \textbf{Avail\_colour}) \bowtie \pi_{\text{colour,prod\#,name,cost}\to\text{cost2}}(\textbf{Product} \bowtie \textbf{Avail\_colour}))))$$

Correct answer because we project down to π_{colour} at end!

State what the query asks in English.

Note that you will get *zero* credit if you use database terms in your answer! (E.g., "Well, the query first *joins* two tables, taking the *projection* of..." does not count!)

List all colours that some product comes in. For each available colour (colour), show the most expensive product by prod# and name that comes in that colour. (In case of tie for most expensive, list all in the tie.)

This is what query has "done" before final projection

2: correct description
1: mostly correct ← least costly in
∅: Otherwise

(e) [2pt] Consider the following SQL query.

```
select distinct Z.A, X.C
from R Z, S X
where Z.B not in (
        select Y.B
        from S Y
        where X.C = Y.C);
```

Write an equivalent R.A. expression for it.

$$(\pi_A(R) \times \pi_C(S)) - \pi_{A,C}(R \bowtie_B S)$$

2: Cross product and minus correct
1: Catches main idea, but a logical flaw
∅: Otherwise

3. (10 points) **Queries in SQL.** *Ask me anything.* EXERCISE

Consider the *Movie* schema in Figure 2 on page 13 for Questions 3a to 3d.

(a) [3pt] Write an SQL query that answers the following.
Report *directors* by p# and name with the movies that they have directed by title, studio, year, and genre.
Order by name, p#, year, studio, title (all *ascending*).

```
Select P.p#, P.name, M.year, M.studio, M.title, M.genre
From Person P, Movie M
Where M.director = P.p#
order by P.name, P.p#, M.year, M.studio, M.title;
```

+1 : Joins Person & Movie correctly
+1 : Correct attr's returned
+1 : Order by

(b) [2pt] Write an SQL query that answers the following.
Report *directors* by p# and name with the number of movies that they have directed as #movies. You may assume that every director has directed some movie.
Order by name, p# (both *ascending*).

```
Select P.p#, P.name, count(*) as #movies
From Person P, Movie M
Where M.director = P.p#
group by P.p#, P.name
order by P.name, P.p#;
```

+1 : Correct ⋈
+1 : Correct group by & aggr.

(c) [3pt] Consider the following SQL query.

```
select  P.p#, P.name,
        ( select count(*)
          from Cast C
          where P.p# = C.p#
        ) as #roles
from Person P;
```

actor

Assume that everyone who has been cast in a movie is an *actor*, and that every *actor* has been in some movie.

Write an SQL query that means the same thing, but that uses *no* nested (sub-) queries.

```
Select P.p#, P.name, count(*) as #roles
From Person P, Cast C
where C.actor = P.p#
group by P.p#, P.name;
```

+1: Correct join

+2: Correct group by & aggr.

(d) [2pt] Consider the following SQL query.

```
select P.name, P.gender, C.role, C.minutes, M.title, M.studio, M.year
from Person P, Cast C, Movie M, Authored A, Person W
where C.actor = P.p#
  and C.title = M.title and C.studio = M.studio and C.year = M.year
  and M.genre = 'SciFi'
  and A.title = M.title and A.year = M.year
  and A.writer = W.p#
  and W.name = 'Hampton Fancher';
```

State what the query asks in English. Note that you will get *zero* credit if you use database terms in your answer! (E.g., "Well, the query first *joins* two tables, taking the *projection* of..." does not count!)

List all cast members & movie details for Sci-Fi movies written (screenplay) by H.F.

List each actor by name, gender, role, and the minutes they appear on screen in that role in a given movie by title, studio, and year such that "Hampton Fancher" was an author of the screenplay of the movie and the movie's genre is "SciFi".

2: Correct 1: Mostly correct 0: Otherwise

4. (10 points) **Query Logic.** *Fascinating.* ANALYSIS

(a) [2pt] Consider the schema

S(B, *C*)
R(A, *B*)
FK (B) refs **S**(B)

What is $|\mathbf{R} \bowtie \mathbf{S}|$? (That is, what is the cardinality of $\mathbf{R} \bowtie \mathbf{S}$?)
Give your answer in the simplest form.

$|R|$

+1 : Correct answer
+1 : Simplest form

(b) [2pt] You execute the following command against the database system for the *Colours* database (schema in Figure 1 on page 13).

```
insert into Item (item#, prod#, cust#, colour, date_sold)
values
    (1729, 23, 13, 'hot pink', '03/14/2019');
```

Is this tuple now guaranteed to have been added to the table **Item**?
Why or why not?

No. 1729 could already be the value of item# of some tuple in Item. this insertion would violate the primary-key constraint. 23 might not exist as a value of prod# in Product, this would violate Item's Foreign key on Product. Etc.

+1: No. +1: A proper explanation

(c) [2pt] Say that "**R**.A <> **S**.B" appears as a predicate in the *where* clause of an SQL query.
To what does the predicate evaluate when **R**.A is *null* and **S**.B is *null*?

unknown. (null compared against anything (even null) evaluates as unknown.)

~~½~~2 : unknown
∅ : Otherwise

(d) [2pt] Consider the *Colours* database (schema in Figure 1 on page 13) and the following SQL query.

A.

```
select distinct C.cust#, A.prod#
from Customer C, Avail_Colours A
where C.fav_colour <> A.colour;
```

State what the query asks in English.

Note that you will get *zero* credit if you use database terms in your answer! (E.g., "Well, the query first *joins* two tables, taking the *projection* of..." does not count!)

List customers paired with products such that the product comes in some colour other than the customer's favourite.

2 : correct
1 : Mostly so
0 : Otherwise

(e) [2pt] Consider the following SQL query.

```
select A, B
from (values
        (3.1416, 'x'),
        (2.7183, 'y')
     ) as R (A, B)
except
select C
from (values
        (3.1416)
     ) as S (C);
```

How does the query evaluate?

It fails with an error; the two sub-queries of the except clause are schema incompatible.

2 : "Fails", with reason
1 : On track, but not fully correct
0 : Otherwise

5. [10pt] **The SQL Language.** *Speaking quite logically.* SHORT ANSWER

Consider the following for Questions 5a, 5b, and 5c.

Dr. Mark Dogfury has written his own SQL database system that he calls **WoofSQL**. Unfortunately, he forgot to implement "distinct", and he forgot to implement "having"! However, he did implement the rest of SQL.

(a) [3pt] Can you still write queries in **WoofSQL** that eliminate duplicate values, despite missing "distinct"?

Why or why not?

Yes. You can list all the attr's you're returning by the select clause in a corresponding group-by clause. This will have the meaning of "distinct".

+1 : Yes

0, 1, 2 : Quality & completeness of explanation

(b) [2pt] Can you still write all the queries that you can in regular, *full* SQL in **WoofSQL**, even though you do not have the *having* clause?

Why or why not?

Yes. You can nest the query you have in mind as a sub-query, but without the having clause. Then, in the new main query's where clause, you can put the predicates from your original having clause.

+1 : yes

0, 1, 2 : Quality & completeness of explanation

(c) [2pt] Dr. Dogfury implemented *distinct* and *having* for **WoofSQL v2**, but somehow the *intersect* operator got left out of **v2**!

Can you still write all the queries that you can in regular, *full* SQL in **WoofSQL v2**, even though you do not have the *intersect* operator?

Why or why not?

Yes.
Do an equijoin between the two tables with the join condition being a conjunct of all the attributes. E.g.
$R(A,B,C) \bowtie_{R.A=S.A \wedge R.B=S.B \wedge R.C=S.C} S(A,B,C)$.

+1: Yes

0, 1, 2: Quality & completeness of explanation

(d) [3pt] Say that you know that the **Product** table in the *Colours* database (schema in Figure 1 on page 13) has 45 tuples in it.

How many tuples are returned by the following query?

```
select prod#, pname
from Product as P,
     (values ('A'), ('B'), ('C'))
       as Q (X);
```

135 tuples are returned. Tables "P" and "Q" are effectively cross-producted. P has 45 tuples, Q has 3.
45 x 3 = 135

3pts +1 → 135 tuples

0, 1, 2: Quality & completeness

2pts: Something very much on track
1pt: Evidence student understood question

EXTRA SPACE

RELAX. TURN IN YOUR TEST. RETURN TO THE WILD! EAT SOME PIE!

MASTER

REFERENCE *(Detach this page for convenience, if you want.)*

Schema for the Colours Database.

Customer	
cust#	PK
cname	
fav_colour	
phone#	

Item	
item#	PK
prod#	FK to **Product**
cust#	FK to **Customer**
colour	
date_sold	

Product	
prod#	PK
pname	
cost	
maker	FK to **Company**

Avail_Colours	
prod#	PK, FK to **Product**
colour	PK

Figure 1: Colours Schema.

Schema for the Movie Database.

Person(p#, *name*, birthdate, *nationality*, *gender*)
Actor(p#, *aguild#*)
FK (p#) refs **Person**
Director(p#, *dguild#*)
FK (p#) refs **Person**
Writer(p#, *wguild#*)
FK (p#) refs **Person**
Studio(name)
ScreenPlay(title, year)
Authored(title, year, writer)
FK (title, year) refs **ScreenPlay**
FK (writer) refs **Writer** (p#)
Movie(title, studio, year, *genre*, *director*, *length*)
FK (studio) refs **Studio** (name)
FK (title, year) refs **ScreenPlay**
FK (director) refs **Director** (p#)
Cast(title, studio, year, role, actor, *minutes*)
FK (title, studio, year) refs **Movie**
FK (actor) refs **Actor** (p#)
Affiliated(director, studio)
FK (director) refs **Director** (p#)
FK (studio) refs **Studio** (name)

Figure 2: Movie Schema.

REFERENCE

The Relational-Algebra Operators.

σ : selection
π : projection
$\bowtie$: join
$\times$: cross product
$\cup$: union
$\cap$: intersection
$-$: difference
ρ : rename