UNIVERSITY OF WATERLOO
MIDTERM EXAMINATION
WINTER TERM 2019

Family Name: ______________________ **First Name:** ______________________

Student Signature: ____

Student Identification Number: ______________________________________

Course Section / Instructor (Please check one): ☐ 001/Ward ☐ 002/Ward

Course Number:	ECE 356
Course Title:	Database Systems
Date of Exam:	Wednesday, Feburary 27th 2019
Time Period:	8:30 PM to 9:45 PM
Duration of Exam:	1 hour and 45 minutes
Number of Exam Pages (including cover sheet):	9 pages / 5 questions
Exam Type:	Closed Book
Additional Materials Allowed:	None

GENERAL INSTRUCTIONS

1. No electronic devices including no calculators.
2. Read the entire questions *carefully* before answering.
3. State any assumptions clearly and whenever needed.
4. No questions permitted. When in doubt, make an assumption and write it down.
5. You are not allowed to leave the examination room in the first 1 hour of the exam.
6. You are not allowed to leave the examination room in the final 10 minutes of the exam.

Illegitimati non carborundum

Question	**Maximum**	**Score**						**Total**
1	25	(a)	(b)	(c)	(d)	(e)	(f)	
2	20	(a)	(b)	(c)	(d)	(e)		
3	20	(a)		(b)		(c)		
4	20							
5	20	(a)			(b)			
Total	105							

Database for the Midterm Exam

The following schema and data will be used for several of the questions on this exam. It is a small database for university course scheduling system, providing details about courses, offerings, instructors, classrooms and departments Neither primary keys nor foreign keys have been specified in this description, and that is deliberate. The table name (relation) is in **bold**.

Instructor (instID, instName, deptID, sessional)
Course (courseID, courseName, deptID, prereqID)
Offering (courseID, section, termCode, roomID, instID, enrollment)
Classroom (roomID, building, room, capacity)
Department (deptID, deptName, faculty)
Explanation:

- **Instructor** defines a unique instructor ID, his/her name, department and sessional status.
- **Course** defines a unique course ID, the course name, the department offering that offers the course, and any prerequisites.
- **Offering** defines an actual offering of a course; the offering comprises the courseID being offered, the section number (integers starting at 1), the term code (the standard UW 4-digit term code: the first three digits define the year (add 1900 to get the year), and the fourth digit is the month in which the course starts (1 (Jan), 5 (May), or 9 (Sept) for Winter, Spring, and Fall offerings, respectively), the room where the section meets, the instructor, and the number of students enrolled.
- **Classroom** defines a unique room ID, together with the building, room number and room capacity.
- **Department** identifies a unique department ID and its name.

A subset of the data is as shown below:

Instructor			
instID int	instName char(10)	deptID char(4)	sessional bool
1	Nelson	ECE	false
3	Jimbo	ECE	false
4	Moe	CS	true
5	Lenny	CS	false

Course			
courseID char(8)	courseName varchar(50)	deptID char(4)	prereqID char(8)
ECE356	Database Systems	ECE	ECE250
ECE358	Computer Networks	ECE	ECE222
ECE390	Engineering Design	ECE	ECE290
MATH117	Calculus 1	MATH	null

Offering					
courseID char(8)	section int	termCode decimal(4)	roomID char(8)	instID int	enrollment int
ECE356	1	1191	E74417	1	64
ECE356	2	1191	E74417	3	123
ECE358	2	1191	E74417	1	123
ECE390	1	1191	E74053	1	102
MATH117	1	1189	RCH111	5	134

Classroom			
roomID char(8)	Building char(4)	Room dec(4)	Capacity int
E74417	E7	4417	138
E74053	E7	4053	144
RCH111	RCH	111	91
RCH101	RCH	101	250

Department		
deptID char(8)	deptName varchar(50)	faculty varchar(50)
ECE	Electrical and Computer Engineering	Engineering
CS	Computer Science	Math
MATH	Math	Math
C&O	Combinatorics and Optimization	Math

1. RA and SQL Comprehension: [25 marks] Considering the database schema and the particular data on page 2, for each of the following queries identify
(i) in a single sentence, what is the query computing?
(ii) what is the output for this particular dataset?
(a) `select count(instID) from Instructor where sessional and deptID='CS';`

3 marks; How many sessional instructors are there in Computer Science?

```
+----------------+
| count(instID)  |
+----------------+
|              1 |
+----------------+
```

(b) $\Pi_{\text{courseID}}\,\sigma_{\text{enrollment}>\text{capacity}}\,\text{Offering} \bowtie_{\text{roomID}} \text{Classroom}$

4 marks; Which courses have an enrollment that exceeds their classroom capacity?

```
+-----------+
| courseID  |
+-----------+
| MATH117   |
+-----------+
```

(c) $\Pi_{\text{instID}}\,\text{Instructor} - \Pi_{\text{instID}}\,\sigma_{\text{termCode}>1179}\,\text{Offering}$

4 marks; Which instructors have not taught a course since 2017?

```
+---------+
| instID  |
+---------+
|       4 |
+---------+
```

(d)
```
select sum(enrollment) from Instructor inner join Offering using (instID)
      where termCode > 1179 and
             courseID not in (select courseID from Course
                                         where Course.deptID = Instructor.deptID);
```

5 marks; How many students have been taught courses where the instructor is not from the same department as the course? (students taking two or more such courses will be counted more than once)

```
+-----------------+
| sum(enrollment) |
+-----------------+
|             134 |
+-----------------+
```

(e) $\Pi_{\text{instName,instID,courseID}}$ Course $\bowtie_{\text{deptID}}$ Instructor $\div$ $\Pi_{\text{instID,courseID}}$ Offering

4 marks; This looks like it is a messed up query that was intended to ask: Which instructors (instName) have taught every course offered by their department? However, that query cannot be done trivially, though a version of it that specifies the department (*e.g.*, which instructors have taught every course offered by the ECE department?") can be done easily, as follows:

$\Pi_{\text{instName,instID,courseID}}$ Offering $\bowtie_{\text{instID}}$ Instructor $\div$ $\Pi_{\text{courseID}}\ \sigma_{\text{deptID}=''\text{ECE}''}$ Course

Instead what it is asking is "if only one instructor from only one department has offered courses, what is the name of that instructor? otherwise, return the empty set"

It returns the empty set, since condition is not true.

```
+----------+
| instName |
+----------+
+----------+
```

(f)
```
select S/T,faculty from
       (select count(instID) as S,faculty from Instructor inner join Department
                           using (deptID) where sessional group by faculty) as SC
       right outer join
       (select count(instID) as T,faculty from Instructor inner join Department
                           using (deptID) group by faculty) as TC
       using(faculty);
```

5 marks; what fraction of instructors are sessional, per faculty?

```
+--------+-------------+
| S/T    | faculty     |
+--------+-------------+
| 0.5000 | Math        |
|   NULL | Engineering |
+--------+-------------+
```

Note "NULL" not "0" for engineering, caused by the right outer join.

2. RA and SQL Query Creation: [20 marks] Considering the database schema and the particular data on page 2, provide the required query in Relational Algebra (RA) or SQL, as specified by the question

5 marks for each; The following are necessarily just examples of how to solve the problem, since there is more than one way to solve each of these.

(a) Create an SQL query that determines what fraction of courses are taught by sessionals.

```
select S/T as FractionTaughtBySessionals from
     (select count(instID) as S from Instructor inner join Offering
                                using (instID) where sessional) as SC,
     (select count(instID) as T from Instructor inner join Offering
                                using (instID)) as TC;
```

(b) Create an RA query to determine which instructors (instID) have only taught courses that are offered by their own department.

Π_{instID} Instructor $\bowtie_{\text{instID}}$ Offering $-$
$\Pi_{\text{instID}}\, \sigma_{\text{Course.deptID} \neq \text{Instructor.deptID}}$ Instructor $\bowtie_{\text{instID}}$ Offering $\bowtie_{\text{courseID}}$ Course

Find all instructors who have offered a course and remove from that set any instructor who has offered a course from outside their department.

(c) If we define enrollment/capacity as the room-usage efficiency usage for a given room, then create an SQL query to compute the average room-usage efficiency by building, listed from most to least efficient.

```
select sum(enrollment)/sum(capacity) as Efficiency,Building from
              Offering inner join Classroom using (roomID)
                 group by Building
                 order by Efficiency desc;
```

Although we allowed `avg(enrollment/capacity)` it is a poor solution based on a bad reading of the question; image a building with one room of capacity 1000 and enrollment 900 and a second room with capacity 100 and enrollment 10; avg(enrollment/capacity) will yield (0.9+0.1)/2 = 0.5, while sum(enrollment)/sum(capacity) will yield 910/1100 = 0.83.

(d) Create an RA query to identify instructors (instID) who have taught both a course and its prerequisite.

$\Pi_{\text{instID}}\, \sigma_{\text{Offering.instID=PrereqOffering.instID}}\ \text{Course} \bowtie_{\text{courseID}}\ \text{Offering} \bowtie_{\text{prereqID=prereqOffering.courseID}}\ \rho_{\text{prereqOffering}}\text{Offering}$

This requires two instances of "Offering" (hence the rename operator), one for the course and one for the prereq of the course; the selection must then be such that the instructor has to the same for both offerings, with the prereq forced to be such by doing a join where courseID in the prereq offering equals the prereqID in the course ofering.

(e) Create an SQL query that determines what fraction of students are taught by sessionals, ordered by faculty.

```
select S/T,faculty from
       (select sum(enrollment) as S,faculty from Instructor
               inner join Offering using (instID)
               inner join Department using (deptID)
            where sessional group by faculty) as SC
       right outer join
       (select sum(enrollment) as T,faculty from Instructor
               inner join Offering using (instID)
               inner join Department using (deptID)
            group by faculty) as TC
       using(faculty) order by faculty;
```

3. Keys and Constraints [20 marks] Primary and Foreign Keys were not specified for the university scheduling schema on page 2, nor were Functional Dependencies.

(a) What are plausible Primary Keys on each of the five relations?

4 marks: 0.5 marks each for the four trivial ones; 2 marks for the one over Offering;

Offering (courseID, section, termCode)
Instructor (instID)
Course (courseID)
Classroom (roomID)
Department (deptID)

(b) What are plausible Foreign Keys for the five relations (be explicit: if there are no foreign keys for a relation, say so; identify what is referenced not just what is referencing)?

14 marks: 1 marks each for the two relations with no foreign keys; 2 marks for each foreign key: one for what it's over and one for what it's referencing Offering;

Table	Attribute(s)	References
Department	None	
Classroom	None	
Instructor	deptID	Department(deptID)
Offering	instID	Instructor(instID)
Offering	courseID	Course(courseID)
Offering	roomID	Classroom(roomID)
Course	deptID	Department(deptID)
Course	prereqID	Course(courseID)

(c) What additional constraints, if any, should be added?

2 marks; any plausible constraint from the following four will earn you the two marks;
Constraints:
1. Department(deptName) should be unique
2. Classroom(Building,RoomNumber) should be unique
3. Offering(enrollment) ≥ 0
4 Classroom(capacity) > 0

Instructor(sessional) should probably be "not NULL" and likewise Course(courseName) and Department(faculty).

Beyond these it becomes a bit sketchy; you might think course names should be unique but that is unlikely to be the case; *e.g.*, ECE453 is the same as SE 465 is the same as CS 447 is the same as ECE 653 is the same as CS 647 (and all are called "Software Testing, Quality Assurance, and Maintenance".

4. Normalization [20 marks] Knowing that each department is part of a faculty (deptID $\rightarrow$ faculty), that courses can have more than one prerequisite, and desiring to be able to do queries based on term (Winter, Spring, Fall) without regard to the particular year (*e.g.*, what courses are offered in the fall term?), what modifications to the schema, if any, are needed to ensure that it is either 3NF or BCNF (your choice)? If there are any new of changed relations, identify them, including any changes or adjustments to primary keys and/or foreign keys, or any other constraints. Explain your reasoning.

deptID $\rightarrow$ faculty has no effect on nromalization as it is implicit already in the Department table.

Likewise, termCode as defined does not preclude doing queries based on the term. For example,

```
select ... from Offering ... where termCode % 10 = 9;
```

will operating on course offerings on the Fall term. It is, in this regard, no different than if we used a datestamp for the term, and generally datestamps are considered acceptable from a normalization perspective. If you wanted to break it into two attributes, though (year, and term), that is fine; simply not deemed necessary. Such a change means the primary key for **Offering** changes from (courseID,section,termCode) to (courseID,section,year,term). A third alternative is to add the attribute "term" to the **Offering** relation. However, doing so makes **Offering** no longer 3NF/BCNF because "termCode $\rightarrow$ term" and so a separate table would be needed: **TermTable**(termCode,term). This will have termCode as a primary key; also, the termCode in **Offering** would then need to be foreign key to TermTable(termCode).

The need for multiple prerequisites means that there is no functional dependency courseID $\rightarrow$ prereqID. That means that prereqID would have to be part of the key for the **Course** relation. However, courseID functionally determines the other attributes. As such, we must split that relation into:
Course (courseID, courseName, deptID)
Prereq (courseID, prereqID)
For the Course relation, deptID is still a foreign key referencing Department(deptID) and courseID is still the primary key.

For the Prereq relation, the primary key would be (courseID,prereqID); there would also be two foreign keys, one over courseID and one over prereqID; both would reference Course(courseID).

5. Indexing, Hashing, and Query Optimization: [20 marks] Considering the university scheduling schema from page 2, we wish to identify where indexes may be of value. Since Primary and Foreign Keys were not specified, and so as to prevent possible errors in the previous questions from affecting this question, in your answers identify *all* indexes, even if those indexes are implied by a primary key or foreign key in your previous answers.

(a) Consider the query:

```
select count(courseID) from Course inner join Department using (deptID)
                                where prereqID is NULL and faculty='Math';
```

Assuming no indexes, what would the execution plan be and what would be the estimated execution time for that plan if the tables are on disk, in contiguous blocks, the number of rows in Course is r_c, the number of blocks in Course is b_c, the number of rows in Department is r_d, the number of blocks in Department is b_d, the time to find a random block on disk is T_s and the time to transfer a block from disk is T_t?

4 marks; Assuming a block nested loop join with Department as outer relation in the join, since it should be smaller than Course (one department offers many courses):

$b_d(T_s + T_t) + b_d(T_s + b_c T_t) = 2b_d T_s + b_d(1 + b_c)T_t$

Each block of the Department table is read and processed against the streaming input from the Course table.

(b) For the query above, identify any indexes over one or more attributes that might potentially improve the query performance. For each index you identify, specify the type of index (B+-tree or Hash or either), whether or not it is a primary or secondary index, if it is a secondary index identify if it is useful if it is an index extension, and justify why the query might benefit from that index.

16 marks: 4 marks for each potential index; Two possible indexes on the join attribute, two possible indexes on each of the predicate attributes. All attributes are categorical data and so either B-tree or Hash would work in all cases.

Course(deptID): secondary index (primary index would be on courseID and table would be organizaed with respect to that); it could be useful as an index extension if there is a prereqID index as an index extension (see below for that use); otherwise it will not. Query could benefit from it by using it for the join: scan the department table and for each record, look up in this index if it exists, and then look up the record to determine if prereqID is null or not.

Department(deptID): primary index; query could benefit from it by scanning the course table and then doing a lookup in this index to find the record for this deptID.

Course(prereqID): secondary index; could be useful as an index extension since we can search down this index to find which courses have a NULL prereqID; we can then use courseIDs from these to intersect with the courseIDs from the join, assuming the Course(deptID) was an index extension, and thus avoid doing a record lookup.

Department(faculty): secondary index; would be useful as an index extension; we search down this index to find the deptID for "math" faculty. Then we can intersect that with the deptID from the course table.

SQL and RA

DDL:

```
create table <table-name> (<attribute> <domain>, ...)
drop table [if exists] <table-name>
alter table <table> add <attribute> <domain>
alter table <table> drop <attribute>
create view <view> as <query expression>

Domain Types: char(n), varchar(n), int, smallint, numeric(p,d)
              bool, real, double precision, float(n), date, time

Integrity     not null, default <V>, primary key (<a1, ..., an>),
Constraints:  unique(<a1, ..., an>),
              foreign key (<am, ..., an>) references (<am', ..., an'>)

Referential   on update <...>, on delete <...>
Actions:      cascade, set null, set default, no action
```

DML:

```
insert into <table> values (...)
update <table> set <attribute = ...> where <predicate>
delete from <table> where <predicate>
select <a1,...,an> from <t1,...,tn> where <predicate>

Options on selection attributes: distinct, as

Options on selection tables: natural join, inner join, outer join
                             <join type> on <predicate> using <attributes>

Set Operations: union, interset, except (use 'all' to retain duplicates)

Aggregate Functions: avg, min, max, sum, count

Grouping: group by, having, order by

Options on predicate: =, !=, <, >, <=, >=, in, not in, exists, not exists,
                      is null, is not null, between, like <string pattern>
```

RA:

σ	select	$\sigma_p(r) = \{t \mid t \in r \land p(t)\}$
Π	project	$\Pi_{A_1,A_2,...A_m}(r) = \{< A_1, A_2, ..., A_m >\mid t \in r =< A^1, A^2, ..., A^n > \land\forall_{i=1}^{m}\exists_j A_i = A^j\}$
$\cup$	union	$r \cup s = \{t \mid t \in r \lor t \in s\}$
$-$	set difference	$r - s = \{t \mid t \in r \land t \notin s\}$
$\times$	cartesian product	$r \times s = \{\text{concat}(r_i, s_j) \mid r_i \in r \land s_j \in s\}$
ρ	rename	$\rho_{s(B_1,B_2,...,B_n)}(r) = \{< B_1, B_2, ..., B_n >\mid t \in r =< A_1, A_2, ..., A_n > \land\forall_i B_i = A_i\}$ (new relation is referred to as s)
$\cap$	intersection	$r \cap s = r - (r - s) = \{t \mid t \in r \land t \in s\}$
$\bowtie$	equi-join	$r \bowtie_{r_i,s_j} s = \sigma_{\forall_{i,j} r_i = s_j}(r \times s)$
$\bowtie$	theta join	$r \bowtie_\theta s = \sigma_\theta(r \times s)$
$\div$	division	$r \div s = \Pi_{R-S}(r) - (\Pi_{R-S}((\Pi_{R-S}(r) \times s) - r)) = \{t_i \in t \mid t \times s \subseteq r\}$
$\leftarrow$	assignment	"tmp $\leftarrow$ expr" is a temporary relation equal to the expression
$⟕$	left outer join	$r ⟕ s = (r \bowtie s) \cup \{rs_i \mid r_i \in (r - \Pi_R(r \bowtie s)) \land rs_i = \text{concat}(r_i, \text{NULL})\}$
$⟖$	right outer join	$r ⟖ s = (r \bowtie s) \cup \{rs_i \mid s_i \in (s - \Pi_S(r \bowtie s)) \land rs_i = \text{concat}(\text{NULL}, s_i)\}$
$⟗$	full outer join	$r ⟗ s = r ⟕ s \cup r ⟖ s$