PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2017 EXAMINATIONS

CSC 343 H1S
Instructor: Horton

Duration — 3 hours

Examination Aids: None

Student Number: |_|_|_|_|_|_|_|_|_|_|

Family Name(s): _____

Given Name(s): _____

# 1:	_____/ 10
# 2:	_____/ 9
# 3:	_____/ 18
# 4:	_____/ 4
# 5:	_____/ 7
# 6:	_____/ 11
# 7:	_____/ 5
# 8:	_____/ 3
# 9:	_____/ 6
# 10:	_____/ 6
# 11:	_____/ 9
# 12:	_____/ 4
# 13:	_____/ 8
TOTAL:	_____/100

The last page of this exam is a reference page. You may tear it off.

Pages 15 and 21 provide extra space for rough work.

A mark of at least 40 out of 100 on this exam is required in order to pass the course.

It's been a real pleasure teaching you this term. Good luck!

Good Luck!

CONT'D...

Question 1. [10 MARKS]

Below is a schema we used in lecture. A few attributes have been removed to simplify. Recall that a course's cName has a value such as "Introduction to Databases", dept has a value such as "CSC" and cNum has a value such as 343.

Student(sID, surName, firstName)
Course(dept, cNum, cName)
Offering(oID, dept, cNum, term, instructor)
Took(sID, oID, grade)

Offering[dept, cNum] ⊆ Course[dept, cNum]
Took[sID] ⊆ Student[sID]
Took[oID] ⊆ Offering[oID]

Part (a) [1 MARK]

no 2 offering with same instructor, different department, one of them 100 level

What rule does this integrity constraint enforce? Check one best answer.

$$\sigma_{O1.dept \neq O2.dept \wedge O1.instructor = O2.instructor \wedge O2.cNum < 200}(\rho_{O1}\, Offering \times \rho_{O2}\, Offering) = \emptyset$$

- ☐ An instructor can't teach for two different departments unless one of the courses is first-year.
- ☐ An instructor can't teach for two different departments unless both of the courses is first-year.
- ☐ An instructor can't teach for two different departments unless neither of the courses is first-year.
- ☐ An instructor *can* teach for two different departments as long as one of the courses is first-year.

Part (b) [1 MARK]

The following integrity constraint is intended to enforce the rule that CSC490 can only be offered in terms when CSC454 is also offered. But the algebra is incorrect. Make the smallest change that will fix the algebra.

$A(term) := \Pi_{term}(\sigma_{dept="CSC" \wedge cNum=490}\, Offering)$
$B(term) := \Pi_{term}(\sigma_{dept="CSC" \wedge cNum=454}\, Offering)$
$B - A = \emptyset$

A-B is emptyset

Part (c) [8 MARKS]

On the next page, write a query in relational algebra that finds the SID of each student who (a) was in every course offering that instructor "Truong" taught in term "Fall16", and (b) in at least one of those offerings, had the highest grade. Use only the basic operators $\Pi, \sigma, \bowtie, \times, \cap, \cup, -, \rho, :=$.

You must define these two intermediate relations:

- Every(SID): the SID of students who were in every offering by Truong in Fall16, *i.e.*, who meet condition (a).
- Max(OID, grade): OID was taught by Truong in Fall16 and grade was the highest given in that offering.

Of course, add other intermediate relations as appropriate.

Relational algebra solution goes here. Continue your answer on the reverse if needed.

Continue your answer here if needed:

Question 2. [9 MARKS]

Part (a) [1 MARK]

This SQL query runs without error:

```
SELECT count(stuff), count(*), count(distinct stuff)
FROM StuffAndNonsense;
```

Put each of the expressions from its SELECT clause in the right spot to complete the inequality below:

count(distinct stuff) ≤ count(stuff) ≤ count(*)

Part (b) [1 MARK]

What is the precise, mathematical meaning of $PQ \rightarrow R$? Hint: There is at least one quantifier.

for all tuples t_1 t_2
(t_1[p]==t_2[p] and t_2[Q]==t_2[Q]) -> t_1[R]==t_2[R]

Part (c) [1 MARK]

Is it possible that a relation with attributes ABCDE has two keys: ACE, and B? Yes No

Part (d) [4 MARKS]

Suppose we have the tables Hansel(one, two) with 10 rows, and Gretel(three, four) with 15 rows. Fill in the table to show the minimum and the maximum possible number of rows in the result of each kind of SQL join.

SQL join	minimum number of rows	maximum number of rows
`Hansel JOIN Gretel ON one = four`	0	150
`Hansel NATURAL JOIN Gretel`	150, 150 natural product if no attr agree is cartesian product	
`Hansel LEFT JOIN Gretel ON one = four`	10	150
`Hansel FULL JOIN Gretel ON one = four`	max(10,15) = 15	150

idea is min happens when table with less cardinality has 1-1 agreement with the larger table, resulting in 1 row in the joined table. The rest of dangling tuple is preserved by padding null

Part (e) [2 MARKS]

Which of the following are true about assertions in SQL?

Assertions are computationally expensive.	True	False
Most DBMSs support assertions.	True	False
Assertions cannot express constraints that hold across tables.	True	False
Any assertion can be expressed instead as a reaction policy.	True	False

reaction policy works on foreign keys only; assertion is more general

Question 3. [18 MARKS]

Now we turn to SQL. This question uses the schema from Assignment 1 (see the reference page at the end of the test). I have added one new table called `Charge`.

Important: You may use the view defined in any subquestion (even if you didnt solve it) when solving other subquestions. Additional views are welcome.

There is much more space for each answer that you will need.

Part (a) [3 MARKS]

Recall the `Subcategory` relation. We will say that if a is a subcategory of b, then b is the supercategory or "supertype" of a. Define a view called `ExtendedType` that, for each item, finds the supertype of its type (or NULL if it doesn't have one). Your result must have the form below (note the column names).

```
 iid | type | supertype
-----+------+----------
```

idea is simply join item and subcategory table

```
create view ExtendedType as
select i1.iid as iid, i1.type as type, i2.type as supertype
from item i1 left join subcategory s on i1.iid=s.a
                join item i2 on i2.iid=s.b
```

Part (b) [6 MARKS]

Define a view called `Popular` that, for each year, reports the ID of the item(s) that were ordered in the highest total quantity (summing across all orders that year). Your result must have the form below (note the column names). **Tip:** If `blah` is a `date`, you can get its year as follows: `extract(year from blah)`.

```
 year | iid  | totalquantity
------+------+---------------
```

```
create view Popular as
select
    extract(year from o.date) as year, l.iid as iid, sum(l.quantity) as totalquantity
from
    orderr o join lineitem l on o.oid=l.oid
group by
    extract(year from o.date), l.iid
having sum(l.quantity) >= ALL (    — filtered item total quantity >= all item quantity sold in that year
    select sum(l2.quantity)
    from orderr o2 join lineitem l2 on o2.oid=l2.oid
    where extract(year from o2.date)=extract(year from o.date)
    group by extract(year from o2.date), o2.iid
)
```

Part (c) [5 MARKS]

Write a SQL query that finds the ID, type and supertype of any item that has been "popular" (ordered in the highest quantity) in at least ten different years, but has never been popular since 2010. Your result must have the form below (note the column names).

```
 iid | type | supertype
-----+------+----------
```

```
create view VeryPopular as
select iid
from Popular p
group by p.iid
having count(p.year) >= 10
```

```
create view NotPopularRecently
(select iid from item)
except
(select iid from Popular where year >= 2010)
```

```
select v.iid as iid, type, supertype
from VeryPopular v join NotPopularRecently n on v.iid=n.iid
                   join ExtendedType e on v.iid=e.iid
```

Part (d) [4 MARKS]

Write SQL code that adds a row to table charge for each order in the database, showing the total cost of the order. The table is defined in the reference page at the end of the exam and has this column structure:

```
 cid  | oid  | amount
------+------+--------
```

```
insert into table charge
(
select cid, oid, sum(i.price * l.quantity) as amount
from orderr o join lineitem l on o.oid=l.oid
                join  item i on l.iid=i.iid
group by oid, cid
)
```

Question 4. [4 MARKS]

Suppose I have a file called `nonsense.ddl` containing this:

```
CREATE TABLE X (
   A INT PRIMARY KEY,
   B INT,
   C INT
 FOREIGN KEY (C) REFERENCES Y(D) ON DELETE CASCADE ON UPDATE RESTRICT
);

CREATE TABLE Y (
   D INT PRIMARY KEY,
   E INT,
   F INT,
FOREIGN KEY (F) REFERENCES Z(G) ON DELETE CASCADE ON UPDATE SET NULL
);

CREATE TABLE Z (
   G INT PRIMARY KEY,
   H INT,
   I INT,
   FOREIGN KEY (H) REFERENCES X(A) ON DELETE CASCADE ON UPDATE RESTRICT
);
```

Suppose the tables have been populated as shown below. Modify the data to show the contents of the three tables after this command is executed:

```
DELETE from X WHERE b = 6;
```

delete cascade update restrict

delete cascade update restrict

delete cascade update set null

delete cascade

delete cascade

X:

a	b	c
5	2	9
4	4	5
3	6	1
2	8	9
1	0	6

Y:

d	e	f
1	9	2
2	1	6
6	1	8
4	4	6
5	6	4
9	3	8

Z:

g	h	i
2	3	1
4	3	2
6	5	3
8	1	4

note its always the referred table delete/update that might affect the referring table

Question 5. [7 MARKS]

Below is an excerpt from a JDBC program that operates on this table:

```
create table Guesses (number int primary key,   name text,   guess int,   age int);
```

Complete the code below. to update the the value of guess for any guesses that the person with name who has already made. Each such guess should be set to one more than the biggest guess made by anyone.

Your Java syntax is not important here. Use the reference sheet to find the API of relevant methods. You must use ? placeholders for name and guess in your SQL statement to update guesses.

```
try {
    // Assume that a connection to the database is already stored in "conn", and
    // that "who" already has a value.  You may assume table Guess has at least one row.
    Connection conn;
    String who;
    PreparedStatement ps;
    ResultSet rs;
    int biggest;
```

```
String query = "select max(guess) as maxguess from Guesses;";
ps = conn.prepareStatement(query);
rs = executeQuery(ps);
while(rs.next()) {
   biggest = rs.getInt("maxguess");
}

String update = "update Guesses set guess=? where name=?";
ps = conn.prepareStatement(update);
ps.setInt(1, biggest + 1);
ps.setString(2, who);
ps.executeupdate();
```

```
} catch (SQLException se) { System.out.println("An exception occurred!"); }
```

Question 6. [11 MARKS]

This question also uses the file `data.xml`:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE a SYSTEM "data.dtd">
<a p="hello">
  <b x="1" y="5"/>
  <c n="100">
     <d real="true" value="40">no way</d>
     <d real="false" value="20">yes way</d>
     <d real="false" value="30">possibly</d>
  </c>
  <b x="3" y="1"/>
  <b x="2" y="6"/>
  <c n="52">
     <d real="false" value="25">truly</d>
  </c>
  <c n="50">
     <d real="true" value="10">really</d>
     <d real="true" value="20">actually</d>
  </c>
</a>
```

Note: I have added whitespace to query output in some places to make it easier to read. The whitespace produced by your code and shown in output you trace will not affect your mark.

1. We want a query that will report, for every `c` element, its `n` value and the number of `d` elements inside it. The output should be a sequence of `report` elements as follows:

   ```
   <report size="100" numd="3"/>,
   <report size="52" numd="1"/>,
   <report size="50" numd="2"/>
   ```

   The code below is correct so far. Fill in the two blanks to complete it.

   ```
   for $thing in fn:doc("data.xml")//c
   return
      <report size =                          numd =                          />
   ```

2. The query below is intended to find the value of attribute `n` for every `d` element that is "true", and produce this output:

   ```
   <cat> <dog n="100"/> <dog n="50"/> <dog n="50"/> </cat>
   ```

   But the query doesn't work. It is not even syntactically correct. Make 2 small changes that will fix it.

```
<cat>
   let $document := fn:doc("data.xml")
   for $x in $document//d[/@real = "true"]
   return
      <dog> { $x/parent::c/@n } </dog>
</cat>
```

3. Consider this query:

```
let $document := fn:doc("data.xml")
for $item in $document/a/c/d
where $item/@value > 25
return <list> { $item } </list>
```

It is intended to find every `d` element with a `value` over 25, and produce this output:

```
<list>
   <d real="true" value="40">no way</d>
   <d real="false" value="30">possibly</d>
</list>
```

The query runs but does not produce the correct output. What is its output?

On the code above, make the smallest change(s) that will correct the query.

4. Consider this query:

```
fn:doc("data.xml")//c/d
    [@value =                                          ]
```

It is intended to find, for each `c` element, the `d` element whose `value` is greatest. It should produce this output:

```
<d real="true" value="40">no way</d>,
<d real="false" value="25">truly</d>,
<d real="true" value="20">actually</d>
```

The query so far is correct, but it is missing an expression. For each of the following expressions, circle Yes or No to indicate whether it would correctly complete the query. All are syntactically valid.

`./parent::c/max(d/@value)`	Yes	No
`parent::c/d/max(@value)`	Yes	No
`max(fn:doc("data.xml")//d/@value)`	Yes	No
`./ancestor::c/max(child::d/@value)`	Yes	No

Question 7. [5 MARKS]

Suppose this file, called `data.xml` is valid with respect to its DTD:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE a SYSTEM "data.dtd">
<a p="hello">
  <b x="1" y="5"/>
  <c n="100">
     <d real="true" value="40">no way</d>
     <d real="false" value="20">yes way</d>
     <d real="false" value="30">possibly</d>
  </c>
  <b x="3" y="1"/>
  <b x="2" y="6"/>
  <c n="52">
     <d real="false" value="25">truly</d>
  </c>
  <c n="50">
     <d real="true" value="10">really</d>
     <d real="true" value="20">actually</d>
  </c>
</a>
```

1. For each of the following rules, circle Yes or No to indicate whether it could be part of the DTD.

   https://stackoverflow.com/questions/16198905/dtd-qualifiers-difference-between-placing-them-inside-or-outside-the-parenthes

`<!ELEMENT a (b, c)+>`	Yes	No	b,b,c not allowed
`<!ELEMENT a (b*, c*)*>`	Yes	No	
`<!ELEMENT b x CDATA #REQUIRED>` should be ATTLIST here	Yes	No	
`<!ATTLIST c n ID #REQUIRED>`	Yes	No	
`<!ATTLIST d real CDATA #REQUIRED value CDATA #REQUIRED>`	Yes	No	

2. Write a DTD definition for attribute `real` that accepts the above instance document and enforces this rule: The value of attribute `real` must be either `true`, `false`, or `unsure`. If this is not possible, explain why.

   <!ATTLIST d real (true|false|unsure) #REQUIRED>

3. Suppose our DTD includes a rule defining an element called `junk` — we just didn't happen to engage it in this XML file. Write a new DTD rule for element `junk` that enforces the following: A `junk` element must contain three or more `c` elements followed by two or more `b` or `c` elements in any order. If this is not possible, explain why.

   <!ELEMENT junc (c,c,c+,(b|c),(b|c)+)>

[Use the space below for rough work. This page will ***not*** *be marked, unless you clearly indicate the part of your work that you want us to mark, and make a reference to it in the relevant question.]*

Question 8. [3 MARKS]

Suppose relation R with attributes $ABCDE$ has these functional dependencies:

$$A \to CD, \quad C \to EB$$

I have decomposed it into two relations: ACD and BE. Give a concrete example to demonstrate that my decomposition is lossy. Explain your answer.

chase test. let <abcde> be a tuple in the joined table after projecting to ACD, and BE
now we show that such tuple is not in the original table. Assume none of abcde is 1

A B C D E
a 1 c d 1
1 b 1 1 e

if we project this instance to ACD, and BE, then join back,
we would get abcde as a spurious tuple

Question 9. [6 MARKS]

Consider the relation R on attributes $ABCDEF$, with the following functional dependencies:

$$A \to EF, \quad CDF \to E, \quad E \to BCD$$

Suppose we have started performing BCNF decomposition on R, and have decided to split R using the functional dependency $CDF \to E$.

Complete the BCNF decomposition, showing your rough work and justifying each step. Put your final answer where shown on the next page, and include the functional dependencies that project onto the relations in your final decomposition. There will be no marks for a correct answer without the rough work.

Rough work:

1. EBCD E -> BCD
2. EF nothing
3. ACDF A -> CDF

remember to compute closure first when splitting relations
CDF^+ = BCDEF

The final decomposition, including FDs:

Question 10. [6 MARKS]

Consider relation $R(A, B, C, D, E, F)$ with functional dependencies S.

$$S = \{BCDE \rightarrow A, \quad E \rightarrow BC, \quad CD \rightarrow AB, \quad D \rightarrow E, \quad \}$$

Compute a minimal basis for S. Show your rough work, and put your final answer where shown on the next page. There will be no marks for a correct answer without the rough work.

Rough work:

D -> AE
E -> BC

A minimal basis is:

Question 11. [9 MARKS]

Part (a) [2 MARKS]

What is the 3NF property? That is, what rule must a non-trivial functional dependency satisfy if a relation is in 3NF?

Given FD X->Y, either
— X is a superkey
— Y is prime, i.e. is in some key of the relation

Part (b) [1 MARK]

Suppose I have a schema that I generated using the 3NF synthesis algorithm. Will the new schema have a lossless join?

Circle one: YES No

Part (c) [4 MARKS]

Suppose we have a relation with attributes $PQRSTU$, and the following minimal basis:

$$\{R \to PT, \quad Q \to SU, \quad PQ \to T\}$$

Produce a correct schema, according to the 3NF synthesis algorithm. Explain all steps in your answer.

RPT, QSU, PQT, QR

find closure of FD in minimal basis. found none is a superkey for PQRSTU
add a relation whose schema is some key, i.e. QR is a key

Part (d) [2 MARKS]

Suppose we have a relation with attributes $LMNOP$. Circle Yes or No to indicate which of the following could be true.

can always remove M to get a key with fewer attributes

M appears only on the RHSs of the FDs and MNP is a key.	Yes	No
L and M appear only on the LHSs of the FDs and the keys are PMO and MNP.	Yes	No
All attributes appear on both sides and the keys are PMO and MON.	Yes	No
N appears only on the LHSs of the FDs and the keys are NO and NP.	Yes	No
N does not appear in the FDs and the keys are NO and LO.	Yes	No

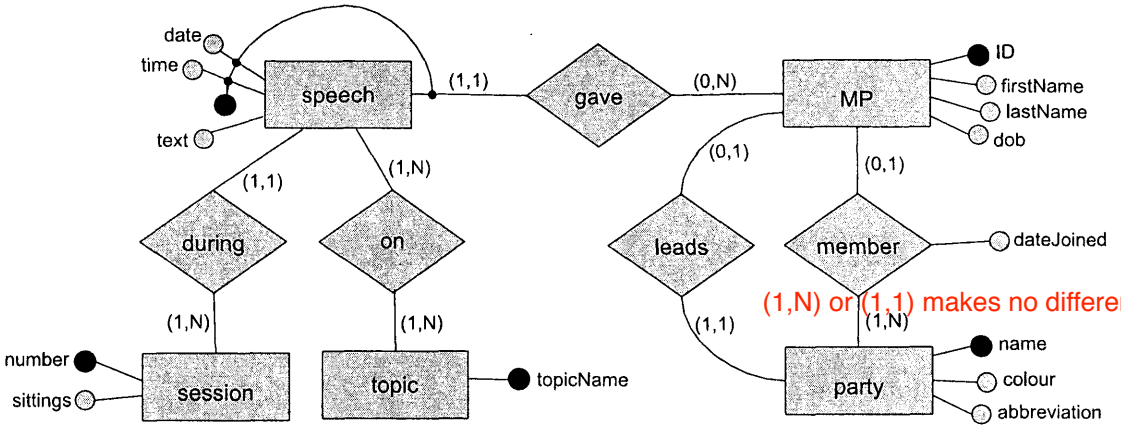
L, M must be in the key

N must be part of key

[Use the space below for rough work. This page will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark, and make a reference to it in the relevant question.]*

Question 12. [4 MARKS]

Below is an Entity-Relationship diagram about members of parliament (MPs) in Canada, the political parties they belong to, and the speeches they give during sessions of parliament.



(1,N) or (1,1) makes no difference here?

The diagram may or may not represent the domain well. Regardless, which of the following is true, according to this Entity-Relationship diagram?

1. This model contains a weak entity set. speech

   True False

2. An MP can give at most one speech.

   True False

3. Two MPs cannot have the same first name and last name unless they belong to different parties.

   True False ID is key

4. Two MPs can give a speech at the same date and time.

   True False key for speech: date, time MP.ID

5. An MP doesn't have to belong to any party.

   True False min=0 optional

6. There can be a party with a leader but no members.

   True False min=1 mandatory has at least 1 member

7. The *gave* relationship is a many-to-one relationship.

   True False

8. The *leads* relationship is a one-to-one relationship.

   True False

Question 13. [8 MARKS]

Translate the Entity-Relationship diagram from the previous question into a relational schema. For each relation, provide its name, attributes and keys. To indicate a key, underline all attributes that are part of the key using a single line. Also include all referential integrity constraints, using relational notation (that is, using $\subseteq$, not SQL notation).

[Use the space below for rough work. This page will **not** *be marked, unless you clearly indicate the part of your work that you want us to mark, and make a reference to it in the relevant question.]*

Schema for the SQL questions

```
-- Only the tables you need for the exam questions are included here.

-- An item that is for sale. IID is the items identification number, type is the type
-- of item it is, such as book, description is a description of the item, manufacturer
-- is the manufacturer of the item, and price is the price of the item.
create table item (
  iid int primary key,
  type text,
  description text,
  manufacturer int references manufacturer(mid),
  price int );

-- A tuple in this relation represents the fact that item type a is a subcategory
-- of item type b.
create table subcategory (
  a text,
  b text,
  primary key (a, b) );

-- An an order by a customer. OID is the order ID, CID is the customer ID, owhen is
-- the date and time on which the order was made, creditCard is the name of the
-- creditCard used for the order, and number is the credit card number.  The table
-- name has a double-r because "order" is a reserved word in SQL.
create table orderr (
  oid int primary key,
  cid int references customer(cid),
  owhen date,      -- Can't be called "when" because that is a reserved word too.
  creditcard text,
  number int );

-- A line item that is part of a particular order. OID is the order ID, IID is
-- the item ID, and quantity indicates how many of the item were ordered.
create table lineitem (
  oid int references orderr(oid),
  iid int references item(iid),
  quantity int,
  primary key (oid, iid)
);

-- The total cost of each order in the database.  CID is the ID of the customer who
-- made the order, IID is the order ID, and amount is the total cost of the order.
create table charge (
  cid int references customer(cid),
  oid int references orderr(oid),
  amount int,
  primary key (cid, oid) );
```

You may tear off this reference page.

Relevant JDBC methods

```
Connection:
   PreparedStatement prepareStatement(String sql)
      Creates a PreparedStatement object for sending parameterized SQL statements to
      the database.

PreparedStatment:
   ResultSet executeQuery()
      Executes the SQL query in this PreparedStatement object and returns the ResultSet
      object generated by the query.
   int executeUpdate()
      Executes the SQL statement in this PreparedStatement object, which must be an SQL
      Data Manipulation Language (DML) statement, such as INSERT, UPDATE or DELETE;
      or an SQL statement that returns nothing, such as a DDL statement.
   void setInt(int parameterIndex, int x)
      Sets the designated parameter to the given Java int value.
      Parameter indices start at 1.
   void setString(int parameterIndex, String x)
      Sets the designated parameter to the given Java String value.
      Parameter indices start at 1.

ResultSet:
   boolean next()
      Moves the cursor forward one row from its current position.
   int getInt(int columnIndex)
      Retrieves the value of the designated column in the current row of this ResultSet
      object as an int in the Java programming language.  Column indices start at 1.
   int getInt(String columnLabel)
      Retrieves the value of the designated column in the current row of this ResultSet
      object as an int in the Java programming language.
   String getString(int columnIndex)
      Retrieves the value of the designated column in the current row of this ResultSet
      object as a String in the Java programming language.  Column indices start at 1.
   String getString(String columnLabel)
      Retrieves the value of the designated column in the current row of this ResultSet
      object as a String in the Java programming language.
```