CS 186/286 Fall 2017 Midterm 2

- Do not turn this page until instructed to start the exam.
- You should receive 1 single-sided *answer sheet* and a 16-page *exam packet*.
- All answers should be written on the answer sheet. The exam packet will be collected but not graded.
- You have *80 minutes* to complete the midterm.
- The midterm has *6 questions*, each with multiple parts.
- For each question, place only your *final answer* on the answer sheet; do not show work.
- Use the blank spaces in your exam for scratch paper.
- You are allowed one 8.5" $\times$ 11" double-sided page of notes.
- No electronic devices are allowed.

1 Relational Algebra

For the following relational algebra questions, please use the schema defined below.

Students(sid, sname, year)
Companies(cid, cname, valuation)
Recruitment(sid, cid, position, salary, status)

For each question, indicate which of the expressions gives the desired output. Note that the following table abbreviations are being used: Students → S, Companies → C, Recruitment → R. Also note that some of the expressions are invalid, in which case they should definitely not be marked in your answer.

1. (2 points) Find the names of all students who have received at least one recruitment offer. Note that if a student has received an offer, the *status* field in the *Recruitment* table will be the text "offer". *Mark all that apply.*

   A. $\pi_{sname}(\sigma_{status=``offer''}(S \bowtie R))$

   B. $\pi_{sname}(S \bowtie (\sigma_{status=``offer''}(R)))$

   C. $\sigma_{status=``offer''}((\pi_{sname}(S)) \bowtie R)$

   D. $\pi_{sname}(\sigma_{status=``offer''}(S \bowtie (\pi_{sid}(R))))$

   Solution: A, B

2. (2 points) Find the names of all students who have not received an offer from any company. *Mark all that apply.*

   A. $\pi_{sname}((\sigma_{status=``offer''}(\pi_{sid}(S) - \pi_{sid}(R))) \bowtie S)$

   B. $\pi_{sname}((\pi_{sid}(S) - \pi_{sid}(\sigma_{status=``offer''}(R))) \bowtie S)$

   C. $\pi_{sname}(S) - \pi_{sname}(\sigma_{status=``offer''}(R \bowtie S))$

   D. $\sigma_{status=``offer''}((\pi_{sname}(\pi_{sid}(S) - \pi_{sid}(R))) \bowtie S)$

   Solution: B

3. (2 points) For every record in Recruitment, output the name of the student, name of the company he/she is being recruited for, and the position the student is being recruited for. *Mark all that apply.*

   A. $(\pi_{sname}(S)) \bowtie (\pi_{cname}(C)) \bowtie (\pi_{position}(R))$

   B. $\pi_{sname,cname,position}(S \bowtie C \bowtie R)$

   C. $\pi_{sname,cname,position}((\pi_{sid,sname}(S)) \bowtie (\pi_{cid,cname}(C)) \bowtie (\pi_{position}(R)))$

   D. $\pi_{sname,cname,position}(S \bowtie C \bowtie (\pi_{sid,cid,position}(R)))$

   Solution: B, D

4. (1.5 points) Apple is an American computing company. Orange is a European telecommunications company. Consider the following two expressions:

$\rho(Apples(1 \rightarrow sid), \pi_{sid}(\sigma_{cname='Apple'}(C \bowtie R)))$
$\rho(Oranges(1 \rightarrow sid), \pi_{sid}(\sigma_{cname='Orange'}(C \bowtie R)))$

Using those expressions, which of the expressions on the answer sheet computes the sids of students being recruited by *both* companies? *Mark all that apply.*

Solution: ✓ $Apples \bowtie Oranges$ ◯ $Apples - Oranges$ ✓ $Apples \cap Oranges$ ◯ $Apples \cup Oranges$ ✓ $Apples - (Apples - Oranges)$ ◯ $Oranges - (Apples - Oranges)$

2 Joins

2.1 T/F Questions

1. (2 points) Mark the statements that are true.

   A. Grace Hash Join will always perform better than Sort Merge Join

   B. It is possible that a hash function will not partition data evenly during the building phase.

   C. If one of the joining relations fits in RAM, Naive Hash Join can outperform Grace Hash Join.

   D. A two-pass Grace Hash join requires memory equivalent to the square root of the larger input relation.

Solution: F,T,T,F

2.2 Young Justice

For the following questions in this section, assume that we are streaming our query output to a terminal. (Do not consider the cost of writing the final output)

```
CREATE TABLE JusticeLeague {
    member_id INTEGER PRIMARY KEY,
    code_name CHAR(33),
    birth_planet CHAR(20),
    power_level INTEGER,
}

CREATE TABLE Teaches {
    member_id INTEGER PRIMARY KEY,
    teacher_id INTEGER REFERENCES JusticeLeague
    since DATE
}
```

We assume that

- JusticeLeague has [J] = 100 pages
- Teaches has $[T]$ = 200 pages
- Buffer size = 12 pages
- Half the JusticeLeague records have `birth_planet = 'Earth'`

2. (2 points) What is the best possible I/O cost of using Page Nested Loops Join to perform a natural join of Teaches and JusticeLeague? Consider both join orders!

Solution:

[J] + [T][J] = 100 + 100*200 = 20100 I/Os $\leftarrow$ This is the optimal.

[T] + [T][J] = 200 + 100*200 = 20200 I/Os

3. (3 points) Consider the following query:

```
-- Find all members who have a earth-born teacher.
SELECT T.member_id
  FROM JusticeLeague J, Teaches T
 WHERE T.teacher_id = J.member_id
   AND J.birth_planet = 'Earth';
```

Given the fact that half of the JusticeLeague members are earth-born, what is the best possible I/O cost to evaluate the query using Block Nested Loop Join? Assume you evaluate the selection operator *on the fly* (no materialization). Consider both $scan_T \bowtie_{BNLJ} (\sigma(scan_J))$ and $(\sigma(scan_J)) \bowtie_{BNLJ} scan_T$.

Solution: Notice we will read in [J] amount of JusticeLeague no matter what. However, if we were to evaluate J.birth_planet='Earth' before performing the join, we will only need to save half of the original number of pages into the buffer.

[J] + 1/2 * [T][J]/(10) = 100 + 100*200/(2*10) = 1100 I/Os ⇐ This is the optimal.

[T] + [T][J]/(10) = 200 + 10*200 = 2200 I/Os

4. (3 points) What is the optimal I/O cost to evaluate the previous query using Grace Hash Join, again considering both join orders with selection on the fly?

Solution:

- Pass 1 read: [J] + [T]
- Pass 1 write partitions: [J]/2 + [T]
- Pass 2 build: [J]/2
- Pass 2 probe: [T]
- Total = 2[J] + 3[T] = 200 + 600 = 800 I/O

3 Parallel Query Evaluation

A local animal shelter that houses dogs, cats, and ducks maintains a database of their animals in the following relations:

Dogs(dog_id, name, birthyear)
Cats(cat_id, name, birthyear)
Ducks(duck_id, name_id, birthyear)
Names(name_id, name)

These relations are distributed across 3 different machines as follows:

- Dogs is stored solely on Machine 1.
- Cats is *range*-partitioned on birthyear over Machines 2 and 3.
- Ducks is *hash*-partitioned on name_id over Machines 1, 2, and 3.
- Names is *round-robin* partitioned over Machines 1, 2, and 3.

The network our database uses supports only "point-to-point" ("unicast") messages. That is, for Machine 1 to send the same message m of b bytes to each of Machines 2 and 3, it must send m to Machine 2, and then separately send m to Machine 3; a total of $2b$ bytes sent.

The following questions consider the following SQL query to get the names of all dogs, cats, and ducks:

```
1         (SELECT name FROM Dogs
2           UNION ALL
3          SELECT name FROM Cats)
4      UNION
5         (SELECT name
6            FROM Names INNER JOIN Ducks
7              ON Ducks.name_id = Names.name_id)
```

1. (2 points) Which of the following operations can be pipeline breakers in this query? Assume that we have very little memory relative to the amount of data. Mark all that apply.

   A. Full Table Scan on Dogs (line 1)

   B. Parallel Hash Join (lines 6, 7)

   C. `UNION ALL` of Dogs and Cats (line 2)

   D. `UNION` of the two subqueries (line 4)

   Solution: The union between the two subqueries is a pipeline breaker: UNION filters out duplicates, and because we are memory-constrained, we cannot filter duplicates by keeping track of previously seen values, and must resort to external sorting or hashing.

   The parallel hash join is a pipeline breaker: we have little memory and therefore a symmetric hash join will not work and we must first partition/hash our data locally, which means reading in everything before returning a tuple.

   The union between Dogs and Cats is not a pipeline breaker since UNION ALL does not filter duplicates, so we can just immediately yield any record obtained from scanning either Dogs or Cats.

   The full table scans are not pipeline breakers, since they yield records immediately and do not require everything to be read in first.

2. (1 point) Using parallel Grace hash join, what is the lowest network cost (the number of bytes sent over the network by any node) possible while performing the join of Names and Ducks?

[N] denotes the number of pages in Names and [D] denotes the number of pages in Ducks. Each page is 4KB.

A. [N] * 4KB

B. [D] * 4KB

C. $\frac{1}{3}$ ([N] + [D]) * 4KB

D. $\frac{2}{3}$ ([N] + [D]) * 4KB

E. ([N] + [D]) * 4KB

F. $\frac{2}{3}$ [N] * 4KB

G. None of the above

Solution: The join between Names and Ducks is a hash join with asymmetric shuffle, since Ducks is already partitioned over the join key (name_id). We therefore do not need to partition Ducks. The network cost of shuffling Names over the machines is $\frac{2}{3}$ * [N] * 4KB, since we have to send $\frac{2}{3}$ of the entire relation over the network once, and since we only send data over the network while shuffling, this is the total network cost. The $\frac{2}{3}$ factor comes from the fact that a third of all the data will be partitioned to the machine it is already on (since we have 3 machines), and therefore is not sent over the network.

3. (0.5 points) True or False? There is an extra disk I/O cost for parallel Grace Hash Join due to repartitioning (shuffling) the data across machines.

Solution: False. Tuples are streamed from the full table scans into the local hash join operators on each of the machines, so the repartitioning does not incur any additional disk I/O cost (there is a network cost, as calculated in the previous question, but no additional disk I/O cost).

4. (1 point) Suppose we range-partition cats by birthyear, so that Machine 2 stores cats born before 1950, and Machine 3 stores cats born 1950 or later. Assume both machines perform scans at the same rate. If 40% of cats in our data were born before 1950, how long will it take for a parallel scan of cats to complete?

[C] denotes the number of pages in Cats, and D represents the time taken to perform a single I/O.

A. 1/3 * [C] * D

B. 0.4 * [C] * D

C. 0.6 * [C] * D

D. [C] * D

E. None of the above

Solution: Machine 1 has no pages of Cats and does nothing.

Machine 2 performs just a full table scan on the 0.4 * [C] pages of cats it has.

Machine 3 performs just a full table scan on the 0.6 * [C] pages of cats it has.

The scans Machine 2 and 3 perform run in parallel, so we are finished once the longer scan finishes, which is 0.6 * [C] * r.

5. (2 points) Now suppose we wish to compare cats and dogs with the same name, and perform the following query:

```
SELECT * FROM Cats INNER JOIN Dogs ON Cats.name = Dogs.name
```

Assuming that [Dogs] = 10 pages and [Cats] = 1000 pages, what is the least number of pages that must be transferred across the network to perform this join?

Solution: We can perform a broadcast join since Dogs is a small table. We send all of Dogs to Machines 2 and 3, and then we join Dogs and Cats locally, resulting in a network cost of 2 * [Dogs] = 20 pages.

6. (1.5 points) In the following questions, select the correct choice to fill in the blank.

   (a) (0.5 points) The term ______________captures the improvement in throughput as the parallelism (number of machines) is increased.

   [✓ **A. speedup**]

   B. scaleup

   (b) (0.5 points) The term ______________captures the ability to maintain throughput as both the parallelism and the size of the data are increased.

   A. speedup

   [✓ **B. scaleup**]

   (c) (0.5 points) *Given: 50% of the Ducks in our database all share the same name_id.* Suppose we run a series of experiments, each time increasing the number of machines over which we hash-partition the Ducks table, and measuring the time to complete a parallel scan of Ducks each time. Across experiments, the completion time for parallel scan will be a ______________function of the number of machines used.

   [✓ **A. constant**]

   B. logarithmic

   C. linear

   D. exponential

   Solution: Ducks is *hash*-partitioned on name_id, so a single machine will get every record with the popular name_id. As we increase the number of machines, we remain bottlenecked by the single machine with the popular name_id (which always has to scan half the records), so the completion time is constant.

4 Query Optimization

For the following questions, assume the following:

- The System R assumptions about uniformity and independence from lecture hold
- We use System R defaults when selectivity estimation is not possible
- Primary key IDs are sequential, starting from 1
- Our optimizer does not consider interesting orders

Table Schema	**Records**	**Pages**	**Indices**

`CREATE TABLE Student (`
`sid INTEGER PRIMARY KEY,`
`name VARCHAR(32),`
`major VARCHAR(64)),`
`semesters_completed INTEGER`
`)`
	25,000	500	• Index 1: Clustered(`major`). *There are 130 unique majors*
			• Index 2: Unclustered(`semesters_completed`). *There are **11 unique values** in the range* $\mathbf{[0, 10]}$

`CREATE TABLE Application (`
`sid INTEGER REFERENCES Student,`
`cid INTEGER REFERENCES Company,`
`status TEXT,`
`(sid, cid) PRIMARY KEY`
`)`
	100,000	10,000	• Index 3: Clustered(cid, sid).
			• Given: `status` has ***10 unique values***

`CREATE TABLE Company (`
`cid INTEGER PRIMARY KEY,`
`open_roles  INTEGER)`
`)`
	500	100	• Index 4: Unclustered(cid)
			• Index 5: Clustered(`open_roles`). *There are **500 unique values** in the range* $\mathbf{[1, 500]}$

```
SELECT Student.name, Company.open_roles, Application.referral
  FROM Student, Application, Company
 WHERE Student.sid = Application.sid                       -- (Selectivity 1)
   AND Application.cid = Company.cid                       -- (Selectivity 2)
   AND Student.semesters_completed > 6                     -- (Selectivity 3)
   AND (Student.major='EECS' OR Company.open_roles <= 50)  -- (Selectivity 4)
   AND NOT Application.status = 'limbo'                    -- (Selectivity 5)
ORDER BY Company.open_roles;
```

For the first 5 questions, write the reduction factor for each clause in the answer box as a fully-reduced fraction. For example, 1/2 should be written as:

1 / 2

Note: Answers that are not reduced fractions or are formatted incorrectly will receive 0 points.

1. (0.5 points) `Student.sid = Application.sid`

Solution:

$$\frac{1}{max(25000, 25000)} = \frac{1}{25,000}$$

There are exactly 25000 values in Student.sid, and due to the foreign key, there are at most 25000 values in Application.sid.

2. (0.5 points) `Application.cid = Company.cid`

Solution:

$$\frac{1}{max(500, 500)} = \frac{1}{500}$$

There are exactly 500 values in Company.cid, and due to the foreign key, there are at most 500 values in Application.cid

3. (0.5 points) `Student.semesters_completed > 6`

Solution:

$$\frac{10 - 6}{10 - 0 + 1} = \frac{4}{11}$$

We have a histogram with 11 unique values. Therefore we use the equation for less than or equal to which is (high key - value) / (high key - low key + 1).

4. (1 point) `Student.major = 'EECS' OR Company.open_roles <= 50`

Solution:

$$(\frac{1}{130} + \frac{1}{10}) - (\frac{1}{130} * \frac{1}{10}) = \frac{10}{1300} + \frac{130}{1300} - \frac{1}{1300} = \frac{139}{1300}$$

We can find the selectivity that they are an EECS major by using the equation 1/distinct values. Next, we find the selectivity that open positions are less than or equal to 50 using the equation (v - low key) / ((high key - low key + 1) + (1 / number distinct)). Lastly we combine these two selectivities using S(p1) + S(p2) - S(p1)S(p2) to determine the selectivity of having one or the other.

5. (0.5 points) `NOT Application.status = 'limbo'`

Solution:

$$1 - \frac{1}{10} = \frac{9}{10}$$

Given 10 unique values, the non-negated predicate has selectivity $\frac{1}{10}$, so we can use the equation for NOT which is 1 - selectivity of the predicate.

6. (2.5 points) For each predicate, mark the first pass of Selinger's algorithm that uses its selectivity to estimate output size.
   (a) Selectivity 1: Pass 1, Pass 2, or Pass 3.
   (b) Selectivity 2: Pass 1, Pass 2, or Pass 3.
   (c) Selectivity 3: Pass 1, Pass 2, or Pass 3.
   (d) Selectivity 4: Pass 1, Pass 2, or Pass 3.
   (e) Selectivity 5: Pass 1, Pass 2, or Pass 3.

Solution: Pass 2, Pass 2, Pass 1, Pass 3, Pass 1. Note that (d)—the OR predicate—is over 2 tables that have no associated join predicate, so the selection is postponed along with the cross-product, until after 3-way joins are done.

7. (1 point) Mark the choices for all access plans that would be considered in pass 2 of the Selinger algorithm.
   A. $Student \bowtie Application$ (800 IOs)
   B. $Application \bowtie Student$ (750 IOs)
   C. $Student \bowtie Company$ (470 IOs)
   D. $Company \bowtie Student$ (525 IOs)
   E. $Application \bowtie Company$ (600 IOs)
   F. $Company \bowtie Application$ (575 IOs)

Solution: A, B, E, and F will be considered because they are not cross products.

8. (1 point) Mark the choices from the previous question for all access plans that would be *chosen at the end* of pass 2 of the Selinger algorithm.

Solution: B and F will be chosen because they have the lower cost for joining the two tables tables.

9. (1 point) Mark the choices for all plans that would be considered in pass 3. Note carefully the nested expressions: you may want to draw them out.
   A. $Company \bowtie (Application \bowtie Student)$ (175,000 IOs)
   B. $Company \bowtie (Student \bowtie Application)$ (150,000 IOs)
   C. $Application \bowtie (Company \bowtie Student)$ (155,000 IOs)
   D. $Application \bowtie (Company \bowtie Student)$ (160,000 IOs)
   E. $Student \bowtie (Company \bowtie Application)$ (215,000 IOs)
   F. $(Company \bowtie Application) \bowtie Student$ (180,000 IOs)
   G. $(Application \bowtie Company) \bowtie Student$ (200,000 IOs)
   H. $(Application \bowtie Student) \bowtie Company$ (194,000 IOs)
   I. $(Student \bowtie Application) \bowtie Company$ (195,000 IOs)
   J. $(Student \bowtie Company) \bowtie Application$ (165,000 IOs)

Solution: Considers F, H

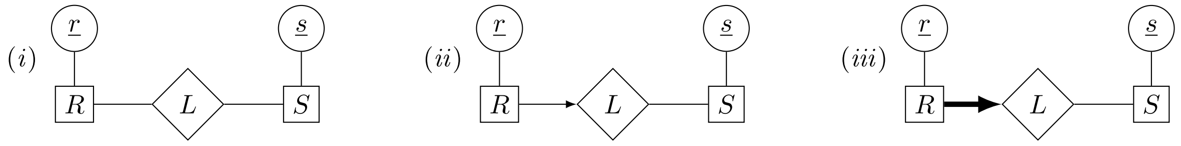
10. (1 point) Mark the choices from the previous question for all plans that would be *chosen at the end of* pass 3.

Solution: Chooses F.

5 Database Design

5.1 ER Diagrams

Consider the following ER diagrams labelled *i*, *ii*, and *iii*.



1. (0.5 points) Which ER diagram do the following SQL statements correspond to? *Note* that there is exactly one correct answer.

```
CREATE TABLE S(s INT, PRIMARY KEY (s));
CREATE TABLE R(r INT, s INT REFERENCES S, PRIMARY KEY (r));
```

Solution: (*ii*). By embedding a reference to `S` in the `R` relation, we ensure that every tuple in `R` can only be linked to at most one tuple in `S`. Because `R.s` can be `NULL`, it is also possible that a tuple in `R` doesn't link any tuple in `S`.

This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

2. (0.5 points) Which ER diagram do the following SQL statements correspond to? *Note* that there is exactly one correct answer.

```
CREATE TABLE R(r INT, PRIMARY KEY (r));
CREATE TABLE S(s INT, PRIMARY KEY (s));
CREATE TABLE L(r INT REFERENCES R, s INT REFERENCES S);
```

Solution: (*i*). By creating a third `L` relation, a tuple in `R` can match many tuples in `S`, and a tuple in `S` can match many tuples in `R`. Moreover, it's possible that tuples in `R` and tuples in `S` don't link with any other tuple.

This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

3. (0.5 points) Which ER diagram do the following SQL statements correspond to? *Note* that there is exactly one correct answer.

```
CREATE TABLE S(s INT, PRIMARY KEY (s));
CREATE TABLE R(r INT, s INT NOT NULL REFERENCES S, PRIMARY KEY (r));
```

Solution: (*iii*). By embedding a reference to `S` in the `R` relation, we ensure that every tuple in `R` can only be linked to at most one tuple in `S`. Because `R.s` is marked `NOT NULL`, every tuple is also guaranteed to be linked to some tuple in `S`.

This question was graded all or nothing. The correct answer received 0.5 points; every other answer received 0 points.

5.2 Functional Dependencies and Normalization

4. (1 point) True or False? For all sets of functional dependencies F and G, $F^+ \cup G^+ = (F \cup G)^+$. That is, closure is *homomorphic.*

Solution: False. Consider a relation with three attributes x, y, and z. Let $X = \{x\}$, $Y = \{y\}$, and $Z = \{z\}$. Let $F = \{X \to Y\}$, and let $G = \{Y \to Z\}$. $X \to Z \notin F^+$ and $X \to Z \notin G^+$, but by transitivity $X \to Z \in (F \cup G)^+$. Thus, $F^+ \cup G^+ \neq (F \cup G)^+$.

This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

5. (2 points) Which of the following functional dependencies are in the closure of $\{T \to UV, TV \to WX, UX \to Y\}$? Mark all that apply.

- $UX \to W$
- $V \to X$
- $T \to X$
- $T \to Y$

Solution: $UX \to W \notin F^+$ and $V \to X \notin F^+$. $T \to UV$ and $TV \to WX$, so $T \to WX$. Thus, by decomposition, $T \to X$. Moreover, $UX \to Y$, so $T \to Y$.

This question was graded as 4 independent true/false questions, each worth 0.5 points. That is, you got 0.5 points for every correct choice that you selected and 0.5 points for every incorrect choice that you did *not* select.

6. (2 points) Given the relation R and functional dependencies above, which of the following attribute sets are *superkeys*? Mark all that apply.

- T
- UV
- TV
- UX

Solution: An attribute set X is a superkey if $X^+ = R$. $T^+ = TUVWXY$, so T is a superkey. $(UV)^+ = UV$, so UV is not a superkey. $TV^+ = TUVWXY$, so it is a superkey, and $UX^+ = UXY$, so UX is not a superkey.

This question was graded as 4 independent true/false questions, each worth 0.5 points. That is, you got 0.5 points for every correct choice that you selected and 0.5 points for every incorrect choice that you did *not* select.

7. (1 point) Given the relation R and functional dependencies above, which of the following attribute sets are *candidate keys*? Mark all that apply.

- T
- UV
- TV
- UX

Solution: An attribute set is a candidate key if it is a minimal superkey. $T^+ = TUVWXY$, and T is minimal, so T is a candidate key. $(UV)^+ = UV$, so UV is not a superkey. $TV^+ = TUVWXY$ but TV is not minimal, so it is a superkey but not a candidate key, and $UX^+ = UXY$, so UX is not a superkey.

This question was graded as 4 independent true/false questions, each worth 0.25 points. That is, you got 0.25 points for every correct choice that you selected and 0.25 points for every incorrect choice that you did *not* select.

8. (1 point) Is R in BCNF?

Solution: No. UX is not a superkey, so the non-trivial dependency $UX \to Y$ violates BCNF.

This question was graded all or nothing. The correct answer received 1 point; every other answer received 0 points.

9. (2 points) Which of the following are lossless-join decompositions of R? Mark all that apply.
   - A. TVW and UXY
   - B. T and $UVWXY$
   - **C. $TUVWX$ and UXY**
   - **D. $TUVWX$ and $TUXY$**

Solution: A and B are not lossless-join decompositions.

C is produced by following the decomposition algorithm presented in class. We use the $UX \to Y$ functional dependency to separate $TUVWXY$ into $TUVWXY - Y$ and $UX \cup Y$. This is a lossless-join decomposition.

D is a lossless join decomposition because both $TUVWX$ and $TUXY$ contain T which is a candidate key.

This question was graded as 4 independent true/false questions, each worth 0.5 points. That is, you got 0.5 points for every correct choice that you selected and 0.5 points for every incorrect choice that you did *not* select.

10. (1 point) Which of the following are lossless-join decompositions of R *into BCNF*? Mark all that apply.
   - A. TVW and UXY
   - B. T and $UVWXY$
   - **C. $TUVWX$ and UXY**
   - D. $TUVWX$ and $TUXY$

Solution: A and B are not lossless-join decompositions.

C is produced by following the decomposition algorithm presented in class. We use the $UX \to Y$ functional dependency to separate $TUVWXY$ into $TUVWXY - Y$ and $UX \cup Y$. This is a lossless-join decomposition. Moreover, both $TUVW$ and UXY are in $BCNF$.

D is a lossless join decomposition because both $TUVWX$ and $TUXY$ contain T which is a candidate key. However, $TUXY$ is not in BCNF because UX is not a superkey in $TUXY$, but $UX \to Y$.

This question was graded as 4 independent true/false questions, each worth 0.25 points. That is, you got 0.25 points for every correct choice that you selected and 0.25 points for every incorrect choice that you did *not* select.