CSE 344 Final Exam

Friday August 18, 2017

Name: ______Solution________ Student Number: ______________________

Problem	Topic	Points
I	Conceptual Design	38
II	BCNF	14
III	Transactions and Serialization	20
IV	Parallel Data Processing	28
Total		**100**

- **Do not open** the test until instructed to do so.
- Please read all instructions carefully. You may ask the instructor clarifying questions during the exam.
- This is a closed book exam, but you are allowed two (double sided) pages of notes.
- Please silence all cell phones and place them off the table.
- There are 4 questions each with multiple parts. If you get stuck on a question move on and come back to it later. Partial solutions will be graded for partial credit.
- You have 60 min to work on this exam.
- If you need, scratch paper is provided at the front of the room. Good Luck!

Part I: E/R Diagrams and Conceptual Design (38 Points)

A) Match each of the following create table statements with the letter that indicates the corresponding entity-relation model from below. (8 points)

1) __B___

```
CREATE TABLE Employee( id PRIMARY KEY, name, office_id UNIQUE
            REFERENCES  Office)
CREATE TABLE Office(id PRIMARY KEY, location)
```

2) __C___

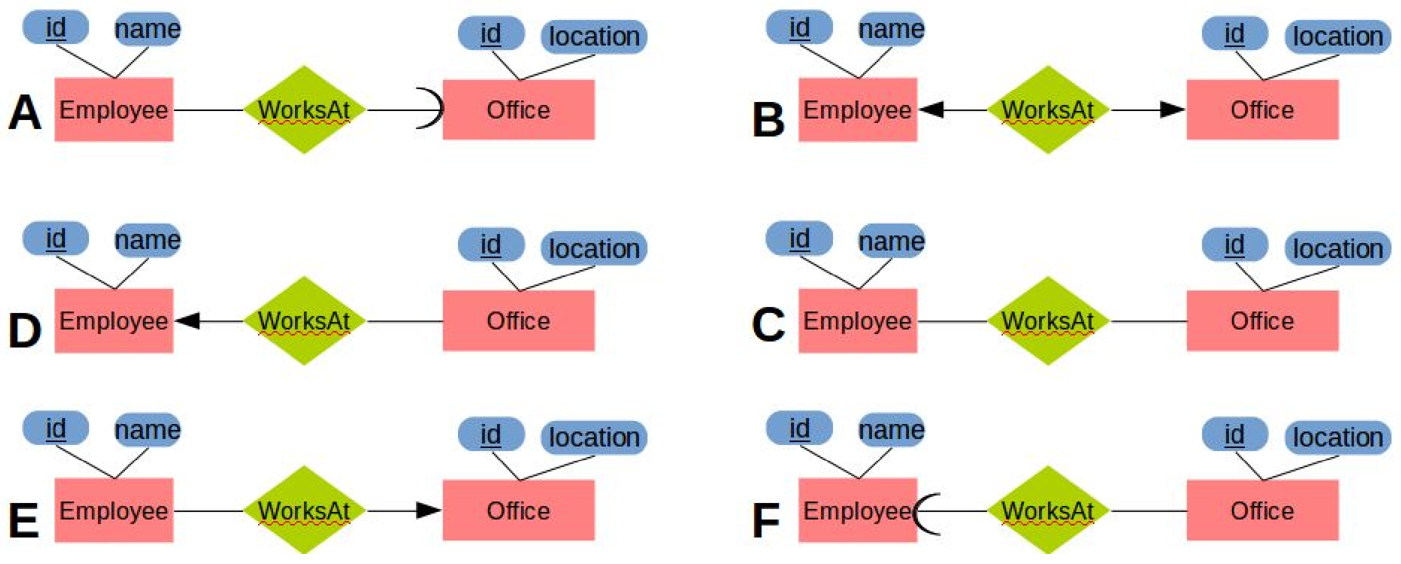
```
CREATE TABLE Employee(id PRIMARY KEY, name)
CREATE TABLE Office(id PRIMARY KEY, location)
CREATE TABLE WorksAt(employee_id REFERENCES Employee ,
           office_id REFERENCES Office)
```

3) __E___

```
CREATE TABLE Employee(id PRIMARY KEY, name, office_id REFERENCES Office)
CREATE TABLE Office(id PRIMARY KEY, location)
```

4) __A____

```
CREATE TABLE Employee(id PRIMARY KEY, name,office_id NOT NULL
           REFERENCES Office)
CREATE TABLE Office(id PRIMARY KEY, location)
```



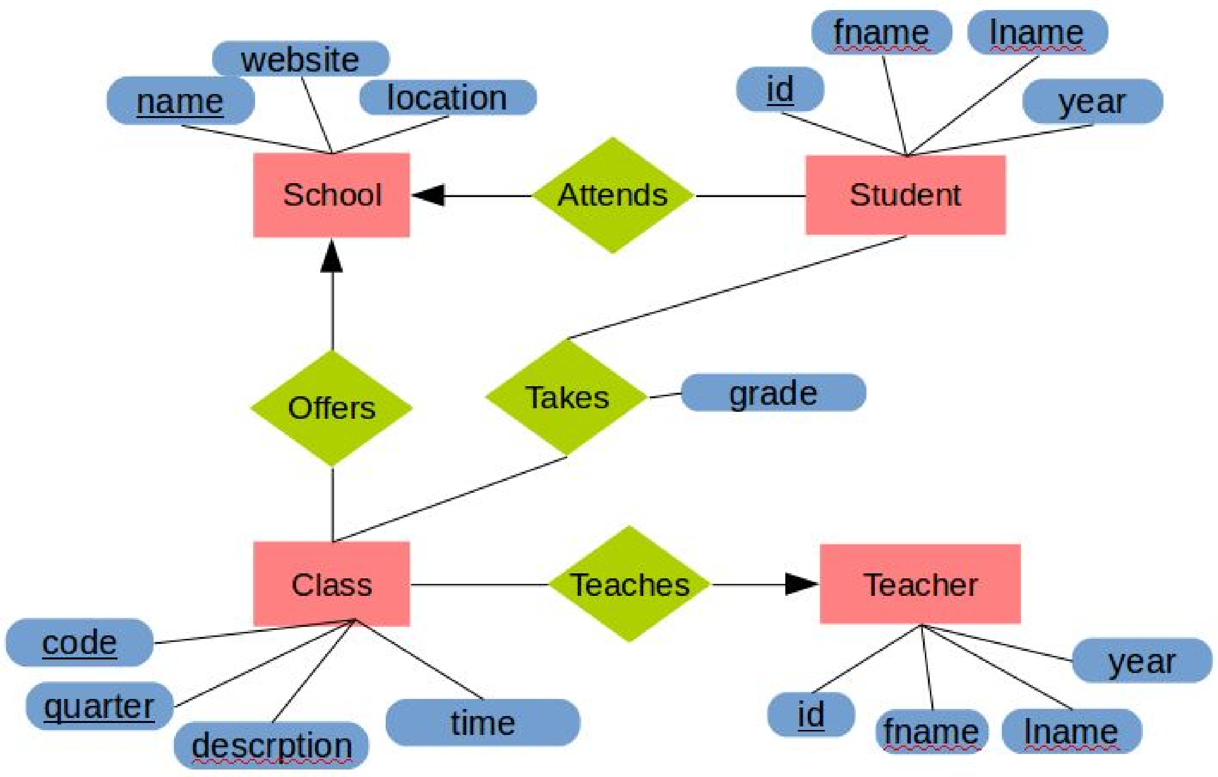
B) Design an E/R Diagram described by the following domain (16 points):

Entities

- A **School** has a unique **name** (key), **location** (address string), and **website**
- A **Student** has attributes **id** (key), **fname**, **lname**, and **year**
- A **Teacher** has a employee **id** (key), **fname**, **lname**, and **salary.**
- A **Class** has a class **code**, **quarter,** and **description, time** where a single class may be offered only once each quarter.

Relationships

- A Student **Attends** only one school but **Takes** many classes. Each class that a Student takes has a **grade** associated with it.
- A **School** offers many **Classes** but a class can only be offered by one school.
- A **Teacher** can **Teach** many classes.



Any constraints not specified could be made more complex. For example making Teacher and Student subclasses of a more general entity was ok. Even though no relationship said that there MUST be a foreign key not null constraints were also accepted.

C) You have implemented the design for part B. However, now additional constraints need to be added. Here are some of the possible types of constraints you could add (6 points)

A. *Key Constraint*: Which type (primary or secondary) and on which attribute.
B. *Attribute Constraint*: say what you would check and on what attribute.
C. *Tuple Constraint*: On which relation and what would you check.
D. *Application Constraint:* Explain why you wouldn't do this in the database.

For each of the following requirements select the most appropriate type of constraint and describe which attributes/tuples it would apply to.

a. An employee's salary must be greater than or equal to 0.

   Attribute constraint on Teacher.salary. (CHECK salary >= 0)

b. A school's website can not be the same as another school's website.

   Secondary key constraint. Unique on School.website.

c. A student may take a maximum of 8 classes in a quarter.

An application level constraint. Make the application responsible for checking that no student has more than 8 classes in a quarter when a new tuple is added to the Takes relationship.

A tuple constraint on Takes. CHECK (NOT EXISTS (SELECT * FROM Takes HAVING count(*) > 8))

D) You are working with a web API that returns JSON about music albums in the following format (8 points):

```
{"albums": [    {"name":"A Hard Day’s Night", "year":1964, "label":"Parlophone",
  band":{"name":"The Beattes","start_year":1960,"end_year":1970},
  "songs":[ {"track":1,"title":"A Hard Day’s Night","length":"2:34"},
    {"track":2,"title"I Should Have Known Better":"length":"2:43"},
    {"track":6,"title":"Tell Me Why","length":”2:09”},
    …. ] },
  {"name":"Abbey Road","year":1969,”label”:”Apple”,
  band":{"name":"The Beattes","start_year":1960,"end_year":1970},
  "songs":[ {"track":1,"title":”Come Together”,"length":”4:20”},
    {"track":2,"title":”Something”,"length":”3:03”},
    {"track":5,"title":”Octopus’s Garden”,"length":”2:51”},
    ….] },   ….
  {"name":"Physical Graffiti","year":1975,”label”:”Swan Song”,
  band":{"name":"Led Zeppelin","start_year":1968,"end_year":1980},
  "songs":[
    {"track":1,"title":”Custard Pie”,"length":”4:15”},
    {"track":2,"title":”The Rover”,"length":”5:39”},
    {"track":10,"title":”Ten Years gone”,"length":”6:34”},
    ….] }  ] }
```

You want to do complex queries on the data and need to create a relational data model. Write the create table statements to hold this JSON data. (Hint: you should have at least 3 tables).

```
CREATE TABLE Band (
        name VARCHAR(100) PRIMARY KEY,
        start_year int NOT NULL,
        end_year int
)

CREATE TABLE Album (
        name VARCHAR(150) PRIMARY KEY,
        year int,
        label VARCHAR(50),
        band VARCHAR(100) REFERENCES Band
)

CREATE TABLE Song (
        track int,
        title VARCHAR(150),
        length VARCHAR(8),
        album VARCHAR(150) REFERENCES Album,
        PRIMARY KEY (album,track)
)
```

Part II BCNF and Functional Dependencies (14 points)

Consider the following relational schema and set of functional dependencies.

R(a, b, c, d, e, f, g) with functional dependencies:

a -> d   g -> bc   e -> a   d -> ef

a) Which of the following are non-trivial implied functional dependencies from the above (circle all that apply). (2 points)

a -> b   [✓ a -> ef]   [✓ d -> a]   d -> d   d -> c   g -> ad

[✓ e -> f]   f -> d   c -> b   e -> b   b -> c   f -> e

b) Compute {d}+, the closure of d. (2 points)

{ d, e, f, a }

c) List two reasons why it is useful to have a set of relations in BCNF. (2 points)

- Reduce data redundancies - BCNF ensures that attributes with the same information are not repeated.
- Remove anomalies - avoid multiple row updates and deletes to keep database consistent

d) List one reason why it might not be useful to have a set of relations in BCNF. (2 points)

- Breaking the relations into a lot of small tables could result in many joins. For example if a table contains the fields of an address even if city implies both state and zipcode it might not make sense to break those out recreating an address would require multiple joins.
- BCNF is always lossless but not always dependency-preserving. If this is an issue it might be acceptable to break BCNF to preserve dependencies.

f) Decompose R into BCNF. Show your work for partial credit. Your answer should consist of a list of table names and attributes and an indication of the keys in each table (underlined attributes). (6 points)

R(a, b, c, d, e, f, g) with functional dependencies:

a -> d   g -> bc   e -> a   d -> ef

```
                      R ( a , b, c, d, e , f, g)
                                  |
                         {a+} = {a,d,e,f}

                                 / \
                    /--------------   ------------- \
              R1 (a , d , e, f)          R2( a, b, c, g)
                                                |
                                        {g+} = {g, b, c}
                                               / \
                                   / -------------   -------------- \
               R21 ( g, b, c )                    R22( g, a )
```

Part III Transactions and Serialization (20 points)
For each of the following schedules, draw the precedence graph and decide if the schedule is conflict serializable. Label each edge of the precedence graph with the data that causes the conflict (e.g. A,B,C ect). If the schedule is serializable, list an equivalent serial schedule of transactions (e.g. T1, T2, T3 - you do not need to list the individual read/write steps). If it is not serializable, briefly explain why not.

a) **r1**(A) **r2**(B) **w3**(A) **r3**(C) **r2**(A) **w1**(B) **r3**(D) **w1**(B) **w3**(D) **w1**(C) (5 points)
Precedence Graph:

$$\begin{tikzcd}
3 \arrow[r, "A"'] \arrow[rr, bend right, "C"'] & 2 \arrow[r, "B"'] & 1 \arrow[ll, bend right, "A"']
\end{tikzcd}$$

Serializable: ____NO____
Serial Schedule or why none exist:

T1 has to read A before T3 writes A, but T3 needs to read C before T1 writes C. This cyce in the dependence graph makes it impossible to create a serial schedule.

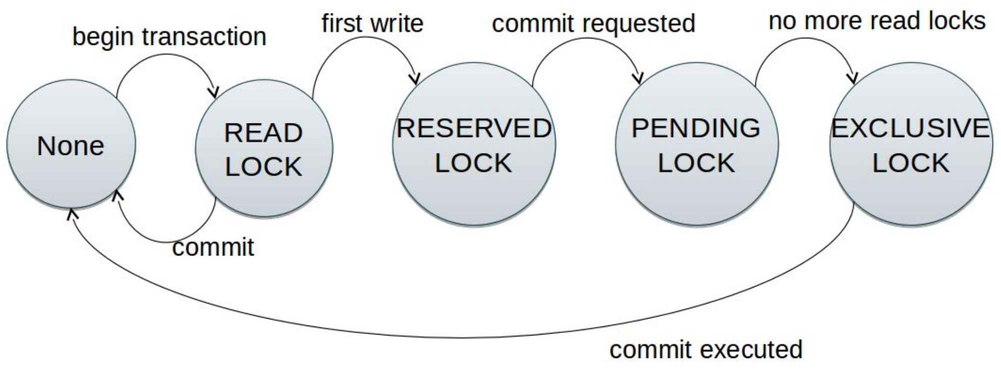
b) **r1**(X) **r3**(Z) **r2**(W) **r4**(Y) **w2**(W) **w2**(X) **w4**(Y) **w3**(Z) **w3**(X) **r1**(Y) (5 points)
Precedence Graph:

$$\begin{tikzcd}
4 \arrow[r, "Y"'] & 1 \arrow[r, "X"'] \arrow[rr, bend left, "X"] & 2 \arrow[r, "X"'] & 3
\end{tikzcd}$$

Serializable: ___Yes____
Serial Schedule or why none exist:

4 , 1, 2, 3

c) SQLite has four different types of locks as illustrated in the state machine below. (10 points)



There are two SQLite transactions (T1 and T2) implemented in the order from the table below.

Time	T1	T2
1	begin transaction	
2		begin transaction
3		select * from A where x = y
A	Indicate below which lock each transaction has. And if T3 can start.	
4	select * from B where z = 10	
5		update A set x = 1 where y = 1
B		
6		commit
C		
7	commit	
D		

For each time slot (A,B,C and D) list which lock T1 and T2 have. Also indicate if a new transaction could start at that point in time.

Yes / No

A) Lock T1: ____READ_________ Lock T2: ____READ____ Start T3: __YES__

B) Lock T1: ____READ_________ Lock T2: __RESERVE__ Start T3: __YES__

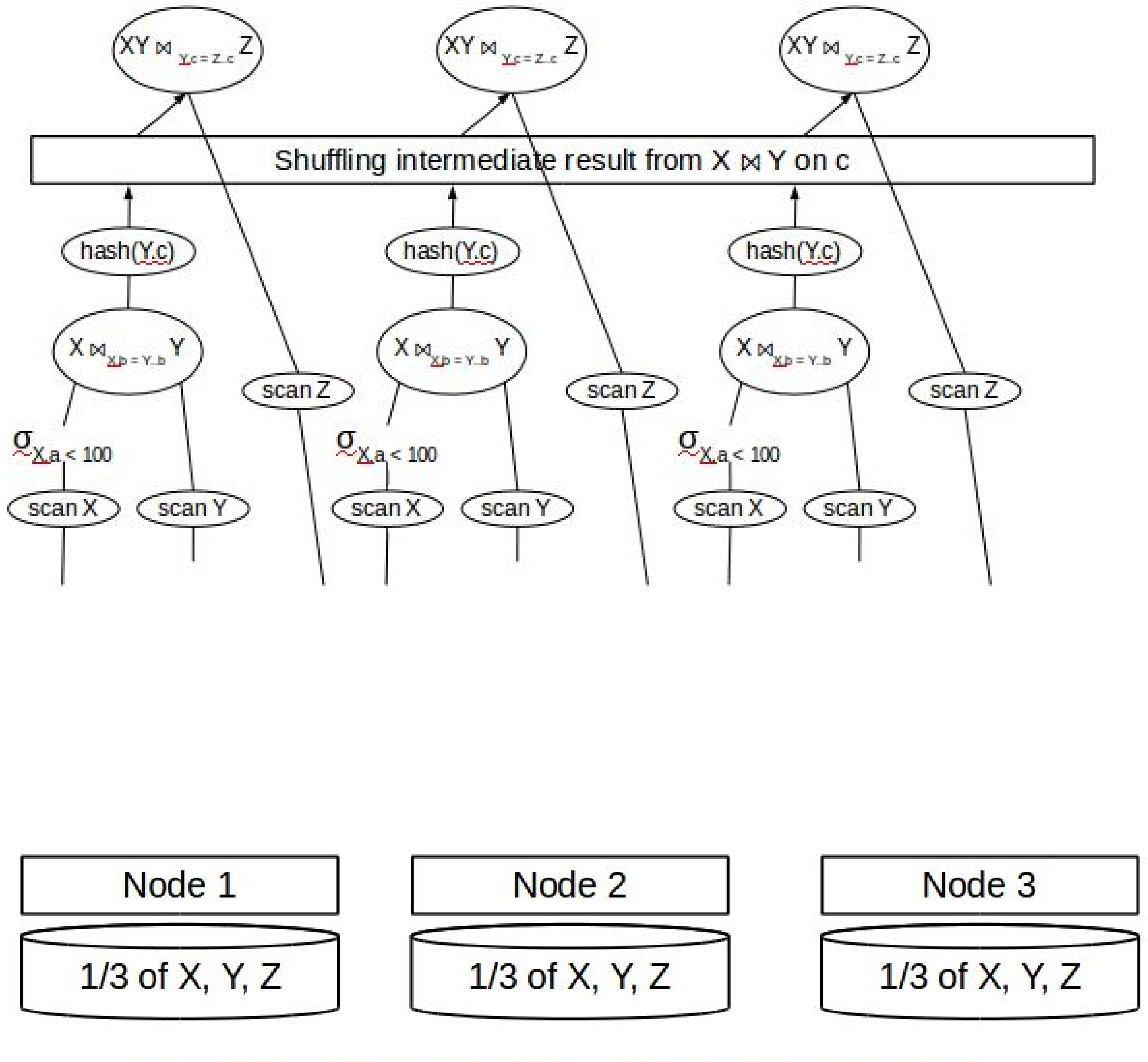
C) Lock T1: ____READ_________ Lock T2: __PENDING__ Start T3: __No__

D) Lock T1: ____None__________ Lock T2: ____None_____ Start T3: __YES__
Or Lock T2: __EXLUSIVE___ Start T3: __No__

Part IV: Parallel Data Processing (28 points)

a) Consider relations X(a,b) , Y(b,c) and Z(c,d). All three are horizontally partitioned across N = 3 machines as shown in the diagram below. Each machine stores approximately ⅓ of the tuples in X,Y, and Z. The tuples in X and Y are hash partitioned on b while Z is hash partitioned on c. Show a relational algebra plan for the following query and how it will be executed across all machines. Use hash-join (shuffle-join) operators and clearly indicate how each re-shuffle operation is performed (make a side note if necessary). (14 points)

SELECT * FROM X, Y, Z WHERE X.b = Y.b AND Z.c = Y.c AND X.a < 100



X and Y partitioned on h(b) Z is partitioned on h(c)

b) You are using Twitter to study the patterns of migratory birds. A large number of volunteer citizen scientist from all over the world assist you by tweeting bird sightings using the hashtag #BirdSurvey. The response has been overwhelming and you now have a collection of over 5 million tweets in JSON format each geotagged and with specific information about the number and type of birds seen.

The tweets are in a large NoSQL document store where the key is the tweetID and the value is the JSON representation of the tweet. For each of the following MapReduce programs give an explanation of what it calculates. Assume that the function getSightings(tweet) returns a list of (bird,count) pairs within that tweets text. (14 points)

1.

```
map(id,tweet):
        total = 0
        for bird, count in getSightings(tweet):
                total += count
        Emit( tweet.country , total )
```

```
reduce(key,values):
        Emit( key, sum(values) )
```

Description of Output:

The Map function emits key value pairs that are the country of the tweet and total number of bird sightings for that tweet. The Reduce function sums up all the sightings for each country.

Result: a mapping of all countries and number of bird sightings.

2.

```
map(id,tweet):
        Emit( tweet.country, tweet.username )
```

```
reduce(key,values):
        unique_values = set(values)
        Emit( key, unique_values.length )
```

Description of Output:

The Map function emits key value pairs that are the country and username of each tweet. The Reduce function counts how many unique users sent in a tweet from each country.

Result: a mapping of all countries and the number of unique users who reported from that country.

c) Write pseudo-code for the map and reduce functions below. The output of the reduce function should answer each questions (roughly equivalent SQL is also given).

I. Find the total number of tweets each username sent.

```
SELECT T.username, count(*) from tweets T group by T.username;
```

```
map(id,tweet):

    Emit( tweet.username , 1 )

reduce(key,values):

    Emit( key, sum(values) )
```

II. The average length of all tweets from each country.

```
SELECT T.country, average( T.text.length ) from Tweets T group by T.country;
```

```
map(id,tweet):

    Emit( tweet.country , tweet.text.length )

reduce(key,values):

    Emit( key , avg(values) )
```