NAME:	**Daniel Carpenter**
STUDENT ID:	**113009743**
GRADED HOMEWORK NUMBER:	**4**
COURSE:	CS/DSA 4513 - DATABASE MANAGEMENT
SECTION:	ONLINE
SEMESTER:	FALL 2021
INSTRUCTOR:	DR. LE GRUENWALD
SCORE:	

Problem 1 *(Exercise 2.14 in Book at pg. 62, using Figure 2.17)*

a) $\prod_{person\_name}(\sigma_{company\_name = \text{“BigBank”}}(employee \bowtie works))$
b) $\prod_{person\_name,\ city}(\sigma_{company\_name = \text{“BigBank”}}(employee \bowtie works))$
c) $\prod_{person\_name,\ street,\ city}(\sigma_{company\_name = \text{“BigBank”} \wedge salary > 10000}(employee \bowtie works))$
d) $\prod_{person\_name}(employee \bowtie_{employee.city = company.city}(works \bowtie company))$

Problem 2

EMPLOYEE (id, name, age, classid, gender, manager, salary)

and a set of functional dependencies:

SetOfFDs = {(classid, id, gender) -> (salary, manager),
name -> (age, id),
id -> name,
manager -> (gender, age, classid, id)}

a) Find ALL candidate keys for the schema EMPLOYEE; show your work.

1. Define the relations that point to ***no*** other relation:
   - age
   - classid
   - gender
   - salary
   - (id, name, age)
2. Define the relations that allow user to find all other attributes, which unveils the candidate keys:
   - **manager:**
     i. manager -> (gender, age, classid, id)
     ii. using manager's relation:
       ✓ (classid, id, gender) -> (salary, manager)
       ✓ id -> name
     iii. Since manager allows us to point to all fields, it is a candidate key.
   - **(classid, id, gender):**
     i. Since (classid, id, gender) -> (manager, salary), and
     ii. manager is a candidate key
     iii. (classid, id, gender) is a candidate key
   - Therefore, manager and (classid, id, gender) total <u>2 candidate keys</u>

b) **Question: For each of the normal forms (1NF, 2NF, 3NF, BCNF), explain in detail why EMPLOYEE satisfies/does not satisfy with respect to the set of functional dependencies SetOfFDs:**

1. 1NF: The EMPLOYEE schema is in 1NF since every attribute is atomic (i.e., only single value, not divisible, no composite value)
2. 2NF: Satisfied by 1NF condition above, but not the dependency condition.
    - Since there are nonprime attributes that are dependent upon multiple attributes, this dissatisfies 2NF.
    - For example, manager is a candidate key, and name depends on manager. However, name also depends on id. Since there are multiple dependencies, this schema is not 2NF.
3. 3NF: Satisfied by 1NF condition above, but not in 2NF, so cannot be in 3NF.
4. BCNF: Satisfied by 1NF condition above, but not in 2NF or 3NF, so cannot be in BCNF.

c) **Use the decomposition algorithm to obtain the lowest normal form that the schema EMPLOYEE does not satisfy with respect to SetOfFDs. Give its complete set of functional dependencies and candidate keys and explain.**

1. 2NF Decomposition shown in the table below, since lowest satisfied form is 1NF:

FDs	Standard Form	Final Closure
manager -> (gender, age, classid, id)	manager -> gender manager -> age manager -> classid manager -> id	Since id gives us name, and name gives us age, *remove*: manager -> age Since (classid, id, gender) -> manager, *remove*: manager -> gender manager -> classid manager -> id Therefore*, no need for this FD now*.
(classid, id, gender) -> (salary, manager)	(classid, id, gender) -> salary (classid, id, gender) -> manager	(classid, id, gender) -> salary (classid, id, gender) -> manager
id -> name	id - > name	Id -> name
name -> (age, id)	name -> age name -> id	name -> age *remove* name -> id

- Final candidate key: (classid, id, gender)
- Functional Dependencies:
  - i. Functional dependencies (Raw)
    - a. (classid, id, gender) -> salary
    - b. (classid, id, gender) -> manager
    - c. id - > name
    - d. name -> age
  - *ii. Final Functional Dependencies in form of minimal basis:*
    - a. (classid, id, gender) -> (salary, manager)
    - b. id - > (name -> age)

d) **Explain in detail why your decomposition obtained in part (c) is/is not loss-less join.**

1. This decomposition is ***NOT*** loss-less since each functional dependency above can access each other to form the EMPLOYEE schema. Please follow the logic below:
    - Given the candidate key (classid, id, gender), you can access -> (salary, manager)
    - Since id is contained in (classid, id, gender), you can use id to access -> name
    - After getting name, you can access -> age
    - Note again that this is a minimal basis, which reduces redundancy while providing a loss-less join.
2. Explanation using loss-less table decomposition *(note random values chosen for example purposes)*:

EMPLOYEE

classid	id	gender	manager	salary	name	age
1	3	M	5	1000	Bob	68
2	4	F	6	10000	Dylan	70

$\pi_{\text{classid, id, gender, manager, salary}}(r)$

classid	id	gender	manager	salary
1	3	M	5	1000
2	4	F	6	10000

$\pi_{\text{id, name, age}}(r)$

id	name	age
3	Bob	68
4	Dylan	70

$\pi_{\text{classid, id, gender, manager, salary}}(r) \bowtie \pi_{\text{id, name, age}}(r)$

classid	id	gender	manager	salary	name	age
1	3	M	5	1000	Bob	68
2	4	F	6	10000	Dylan	70

---

e) **Explain in detail why your decomposition obtained in part (c) is/is not dependency-preserving**

1. Since $\pi_{\text{classid, id, gender, manager, salary}}(r) \cup \pi_{\text{id, name, age}}(r)$ = the original relation, then decomposition is dependency preserving.

Problem 3

Description of Database:

Many attributes associate with any given car. A car will have a make and model. A model will have varying attributes such as the number of doors, if it is an SUV or not, and details about the aesthetic characteristics of the model (attributes simplified for this example). The aesthetic package can have various colors, or even a metallic finish. This database efficiently tracks details of a Car and is in third normal form.

Database Schema Diagrams:

- Car(makeID, makeName, modelID)
- Model(modelID, aestheticPackageID, numDoors, isSUV)
- AestheticPackage(aestheticPackageID, colorID, hasMetallicFinish)
- Color(colorID, colorName)

Functional Dependencies:

- makeID -> (makeName, modelID)
- modelID -> (aestheticPackageID, numDoors, isSUV)
- aestheticPackageID -> (colorID, hasMetallicFinish)
- colorID -> (colorName)

Test for Third Normal Form:

- makeID: superkey
- modelID: superkey
- aestheticPackageID: superkey
- colorID: superkey and is trivial

Each function dependency satisfies one of the below criteria for 3NF, this database is in 3NF.

For each X -> A in F^+ where X is a set of attributes in R and A is a single attribute in R then

- Either X -> A is trivial FD or
- X is a superkey of R or
- A is a prime attribute of R