CSE-3421M Test #1

Design

Sur / Last Name:
Given / First Name:
Student ID:

- **Instructor:** Parke Godfrey
- **Exam Duration:** 75 minutes
- **Term:** Winter 2014

Answer the following questions to the best of your knowledge. Your answers may be brief, but be precise and be careful. The exam is closed-book and closed-notes. Calculators, etc., are fine to use. Write any assumptions you need to make along with your answers, whenever necessary.

There are five major questions, each with parts. Points for each question and sub-question are as indicated. In total, the test is out of 50 points.

If you need additional space for an answer, just indicate clearly where you are continuing.

Marking Box	
1.	/10
2.	/10
3.	/10
4.	/10
5.	/10
Total	/50

1. **Entity-Relationship Modelling.** *Be a model citizen.* (10 points) [Exercise]

The Repair Shop.

Items come into the shop for repair. Each item is of a *type*; e.g., "2003 Honda Civic". An item is identified by an item *id* (item#) *and* its *type*.

An item comes into the shop because it has a *problem*, of course; e.g., "faulty fuel injector". When the item comes in, a *ticket* is issued. The ticket identifies the item, the problem the item has, and the *day* the item comes in.

The shop keeps a catalogue of *problems*. Each problem entry is an official description of a problem for a given *type* of item; e.g., "faulty fuel injector on a 2003 Honda Civic". A problem catalogue entry states how many *hours* such a repair takes. Any ticket should be *valid*; that is, the problem and the item the ticket states should be of the same type.

Mechanics work in the shop. Each mechanic has a unique employee *id* (emp#). A mechanic may be *licensed* in any number of *skills*; e.g., "fuel injector repair". (And, of course, each *skill* may have several mechanics licensed for it.) Record *when* a mechanic first became licensed for a skill. A skill records an *hourly* rate that a licensed mechanic will earn doing a job requiring the skill.

Each catalogue problem requires a specific *skill* for the repair. Only a mechanic with the appropriate skill may be assigned a given ticket to do the repair.

A *repair* records *when* the repair for a ticket is finished, and by which licensed mechanic. A ticket will have at most one repair. (If the same item breaks again in the same way later, it will be brought back to the shop and a *new* ticket is issued.)

You are to devise an entity-relationship (E-R) diagram to model the repair shop described above (for Question 1b on page 3).

In your E-R, do not add any entity unless absolutely needed. You may broaden keys as needed to accommodate the necessary design constraints. Do not forget attributes, and show all keys.

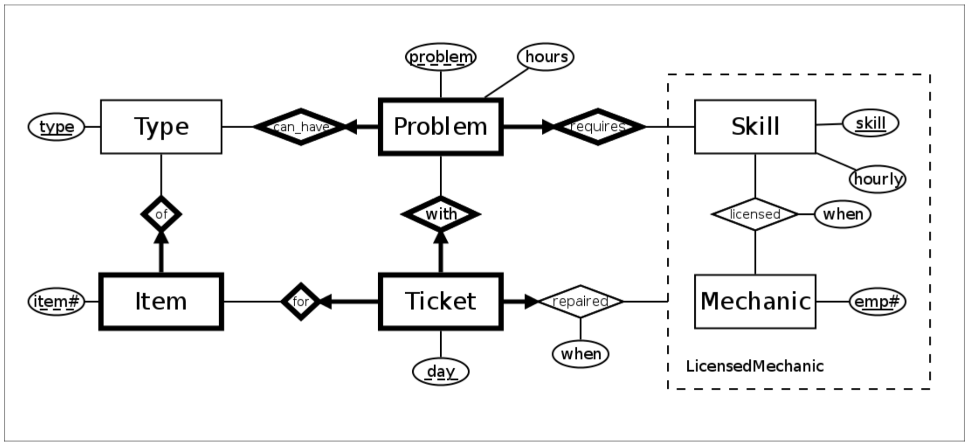
a. (2 points) How should "repair" be modelled in your E-R? E.g., as an *entity*, a *weak entity*, a *many-many relationship*, or a *one-many relationship*.
Briefly explain why.

Repair relates **Ticket** *to* **LicensedMechanic** *(one-many: a* **Ticket** *is related to at most one* **LicensedMechanic** *for when the repair is finished.*
So a relationship.

b. (8 points) Present your E-R.

Components and marking guide:

Enities *(3 pts)*	***M-M Rel-ships &*** ***Aggregates*** *(1 pt)*	***1-M Rel-ships*** *(2 pts)*
a. **Type** *b.* **Item** * *c.* **Problem** * *d.* **Ticket** * *e.* **Skill** *f.* **Mechanic**	*g.* **licensed***:* **skill–mechanic** **LicensedMechanic** *(aggregate)*	*h.* **of***:* **Item** → **Type** *i.* **can_have***:* **Problem**⇒****Type** *j.* **requires***:* **Problem**⇒****Skill** *k.* **with***:* **Ticket**⇒****Problem** *l.* **for***:* **Ticket**⇒****Item** *m.* **repaired***:* **Ticket**⇒****LicMech**
Attributes ***& Keys*** *(1 pt)*	***Weak Entities*** ***(indicated)*** *(1 pt)* *Marked above with "*".*	



2. **Relational Schema.** *You don't choose your relations.* (10 points) [Short Answer]

Assembly(a#,description∗)
Component(c#,name,company,price)
  FK (company) refs **Company**
AuthorizedPart(a#,c#)
  FK (a#) refs **Assembly**
  FK (c#) refs **Component**
Company(company)
Build(a#,b#)
  FK (a#) refs **Assembly**
Part(a#,c#,p#)
  FK (a#, c#) refs **AuthorizedPart**
Composition(a#,b#,c#,p#,qnty)
  FK (a#, b#) refs **Build**
  FK (a#, c#, p#) refs **Part**

Figure 1: Assembly Schema.

The basic schema of a database for tracking machine-shop *builds* of *assemblies* is shown in Figure 1. The attribute qnty indicates the quantity of that **Part** used in the **Build**. The attribute price is the cost of that **Component**.

The underlined attributes indicate a table's primary key (and are, hence, not nullable). Attributes that are appended by an '∗' are not nullable (for example, description∗ in **Assembly**). Foreign keys are indicated by FK.

For Questions 2a to 2c, consider the "reverse engineering" of the relational database schema in Figure 1 to a representative E-R diagram.

a. (2 points) Is **Composition** likely to be an entity or a relationship in an E-R diagram corresponding to the schema in Figure 1? Describe what type of entity it should be (e.g., regular or weak) *or* what type of relationship it should be (e.g., one-many or many-many, binary or ternary).

Likely a relationship as its primary key is the union of its foreign keys to **Build** *and* **Part***. So it can be represented as a many-many binary relationship relating these.*
One could model this as a weak entity too, but there is no need to since there is no partial key component, and nothing needs to relate to it.

b. (2 points) What are the mandatory participations that **Component** is involved in, if any?

None.

c. (2 points) Are there any weak entities? If so, what are they?

Yes:

- **Build** *(on* **Assembly***)*
- **Part** *(on* **AuthorizedPart***)*

d. (2 points) Given this schema design, is it possible to determine the cost of a given **Build**? Assume the cost of a build is the sum of the cost of its parts.
Briefly justify your answer.

Yes. **Composition** *is a many-many relationship that relates* **Build** *and* **Part**. *Each* **Part** *is an* **AuthorizedPart** *which is a* **Component**. *Summing the* **Component***'s price (times qnty from* **Composition***) is the* **Build***'s* price.
Caveat*:* **Component***'s price is nullable; so we are not ensured we compute the actual price.*

e. (2 points) Are we ensured that a **Part** used in a **Build** (**Composition**) is, in fact, of a type of **Component** that is allowed to be used in that type of **Assembly** (**AuthorizedPart**)?
Briefly justify your answer.

Yes, because the same a# (primary key of **assemby***) is used in* **Component***'s foreign key referencing* **Build** *and in its foreign key referencing* **Part**. *So* **Part** *is an* **AuthorizedPart** *for* **Assembly** *a#, and* **Build** *is for* **Assembly** *a#.*

3. **Conceptual to Schema.** *Table that notion!* (10 points) [Exercise]

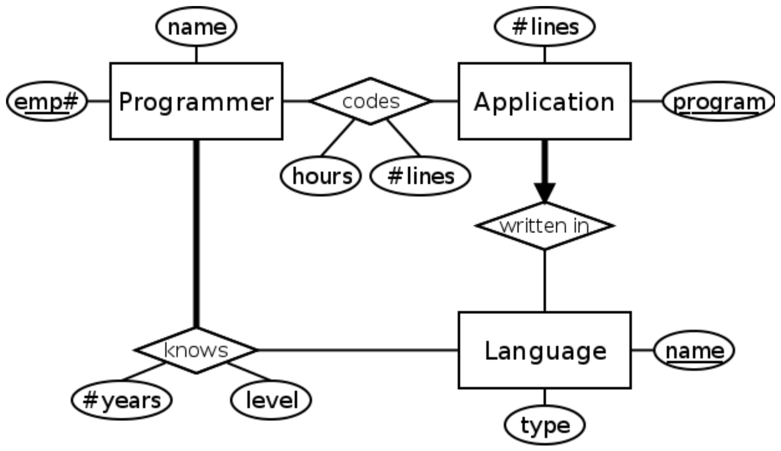


Figure 2: Application E-R.

a. (7 points) Translate the E-R diagram from Figure 2 into a reasonable relational database schema. Use the notation from Figure 1 in Question 2.

Grading guide: 1 pt for each entity and 2 pts that the foreign keys are correct.

- **Programmer**(*<u>emp#</u>*, *name*)
- **Language**(*<u>name</u>*, *type*)
- **Application**(*<u>program</u>*, *#lines*, *name*∗)
  FK (name) refs **Language**
- **Codes**(*<u>emp#</u>*, *<u>program</u>*, *hours*, *#lines*)
  FK (emp#) refs **Programmer**
  FK (program) refs **Application**
- **Knows**(*<u>emp#</u>*, *<u>name</u>*, *#years*, *level*)
  FK (emp#) refs **Programmer**
  FK (name) refs **Language**

b. (3 points) Is there a way to ensure that any **Programmer** who **codes** an **Application** in fact **knows** the **Language** that the **Application** is **written in**?

If not, explain why not.

If so, state a revision to your relational database schema that you provided in Question 3a to ensure this.

One way is as follows.

- *Change* **Application** *to be* weak *on* **Language***, so its primary key is* {*program*, *name*}.
- *Replace the FK in* **Codes** *that references* **Programmer** *with a FK that references* **Knows** *on emp#, name.*

Extra space.

4. **General.** *Eeny meeny miny moe.* (10 points) [Multiple Choice]

Choose one best answer for each. There is no additional penalty for a wrong answer. If you feel clarification for your answer is needed, write a brief clarification next to the question.

a. Functions of a relational database management system include all *except*

- **A.** to ensure integrity constraints are not violated by updates to the data.
- [✓ **B.**] to support general programming functionality through SQL, or another relational query language.
- **C.** to support application programs accessing its databases through SQL.
- **D.** to ensure that the changes of each transaction are committed in entirety or not at all.
- **E.** to support the creation and altering of new databases.

b. Codd's rule of *physical data independence* is that

- **A.** all information in the database is to be represented in one and only one way, namely by values in column positions within rows of tables.
- **B.** all views that are theoretically updateable must be updateable by the system.
- [✓ **C.**] changes that are made to the physical storage representations or access methods must not require changes be made to application programs.
- **D.** changes that are made to tables that do not modify any of the data already stored in the tables must not require changes be made to application programs.
- **E.** data in different tables must not be related.

c. In translating from an entity-relationship (E-R) diagram to a relational schema, one piece of E-R logic that *cannot* be captured by primary keys, uniques, and foreign keys is

- **A.** the weak entity.
- **B.** any ternary relationship.
- **C.** mandatory participation for one-time occurrence (that is, with the arrow).
- [✓ **D.**] mandatory participation for many-time occurrence (that is, without the arrow).
- **E.** aggregation.

d. In an E-R diagram, if one sees a bold line with no arrow between an entity set and relationship set, this means

- [✓ **A.**] every entity in the entity set must participate in the relationship set.
- **B.** every entity in the entity set must appear exactly once in the relationship set.
- **C.** a relationship in the relationship set need not involve an entity from the entity set.
- **D.** the entity set is weak.
- **E.** the entity set *is an* instance of the relationship set.

---

e. Why are the normal forms useful?
  A. They help us find anomalies in the data.
  B. They are just a tool for checking whether our relational design makes sense or not.
  C. If the schema is in BCNF, we are guaranteed that queries will execute faster than if it were not in BCNF.
  [✓ **D.**] By having a relational schema in a given normal form, it guarantees that certain types of data anomalies cannot occur.
  E. They are useless, but earn database consultants lots of money. (Don't tell anyone!)

---

f. If a relation schema is in 3NF and it is known to have just *one* candidate key,
  A. it cannot be in 2NF.
  [✓ **B.**] it must also be in BCNF.
  C. it cannot be in BCNF.
  D. it may be in BCNF or it may not be; there is not enough information to know.
  E. *This is impossible. Any table will always have more than one candidate key.*

For Questions 4g and 4h, consider a relational schema **R** with attributes A, B, C, D, E, F, and G. We know that D never appears on the right-hand side of any functional dependency (FD) in $\mathcal{F}^+$ (the *closure* of the set of FDs, $\mathcal{F}$, known for **R**).

---

g. How many different possibilities are there for what a candidate key for **R** can be?
  A. 1
  B. 6
  C. 7
  [✓ **D.**] 63
  E. 127

---

h. How many different candidate keys could **R** possibly have at the same time?
  A. 6
  B. 7
  [✓ **C.**] 20
  D. 35
  E. 720

```
create table A(a1 int not null primary key,
               a2 int);
create table B(b1 int not null primary key,
               b2 int references A
                   on delete cascade on update no action);
create table C(c1 int not null primary key,
               c2 int references B
                   on delete cascade on update no action);
insert into A values (1,1),(2,1),(3,1),(4,2),(5,2),(6,3);
insert into B values (1,1),(2,1),(3,1),(4,2),(5,2),(6,4);
insert into C values (1,1),(2,1),(3,2),(4,3),(5,4),(6,6);
```

For Questions 4i and 4j, assume the tables **A**, **B**, and **C** were created and populated with data as above, followed by the statement in the question.

Choose the answer that represents what is in table **C** afterwards.

i. `delete from A;`

A. (1,1),(2,1),(3,2),(4,3),(5,4),(6,6)
B. (2,1),(3,2),(4,3),(5,4),(6,6)
C. (4,3),(5,4),(6,6)
D. (5,4),(6,6)
[✓ **E.**] *no tuples*

j. `delete from A where a1 = 1;`

A. (1,1),(2,1),(3,2),(4,3),(5,4),(6,6)
B. (2,1),(3,2),(4,3),(5,4),(6,6)
C. (4,3),(5,4),(6,6)
[✓ **D.**] (5,4),(6,6)
E. *no tuples*

Extra space.

Continue on to Question 5.

5. **Normalization.** *Is this considered normal?* (10 points) [Analysis]

Consider the relation **R** with attributes A, B, C, D, E, and F, and with the functional dependencies

$$\begin{array}{ll} AB \mapsto C & C \mapsto B \\ C \mapsto D & BC \mapsto F \\ D \mapsto E & \end{array}$$

a. (3 points) Consider **R** and the functional dependencies above. What are **R**'s candidate keys?

A only on left in every rule; so must be in every candidate key! E and F are only on the right; so in no candidate key!
$\underline{A}^+ \rightarrow$.
$\underline{AB}^+ \rightarrow$ *CDEF.*
$\underline{AC}^+ \rightarrow$ *BDEF.*
$AD^+ \rightarrow$ *E.*
There are no others to explore that are not supersets of these (that contain A), so done: AB and AC are the two candidate keys.

b. (3 points) Consider again **R** and the functional dependencies (FDs) above. Is **R** in 2NF? Is **R** in 3NF? Is **R** in BCNF?

In each case, if it is not, provide a concrete reason.

$C \rightarrow D$ breaks 2NF since $C \subset AC$ and D is not prime. So not in 3NF nor BCNF for the same reason. (Not in BCNF additionally because $C \rightarrow B$.)

c. (2 points) Consider set $\mathcal{F}$ of the functional dependencies

$$\begin{gathered} \mathsf{AB} \mapsto \mathsf{C} \\ \mathsf{D} \mapsto \mathsf{BE} \\ \mathsf{CE} \mapsto \mathsf{F} \end{gathered}$$

Does the functional dependency $\mathsf{CE} \mapsto \mathsf{F}$ follow from $\mathcal{F}$?
Why or why not?

Well... yes. $\mathsf{CE} \rightarrow \mathsf{F} \in \mathcal{F}$! So of course it follows from $\mathcal{F}$.
(Okay, this was my mistake. I meant to ask, "Does $\mathsf{AD} \rightarrow \mathsf{F}$ follow from $\mathcal{F}$?" And it does because $\mathsf{AD}^+ \rightarrow \mathsf{BEC\underline{F}}$).

---

d. (2 points) Consider relation **S** of four attributes A, B, C, and D, with the functional dependencies

$$\begin{gathered} \mathsf{AB} \mapsto \mathsf{CD} \\ \mathsf{D} \mapsto \mathsf{A} \\ \mathsf{B} \mapsto \mathsf{D} \end{gathered}$$

Is **S** in 3NF? Explain briefly why or why not?

B is not on the right of any FD rule, so it must be in every candidate key.
$\underline{\mathsf{B}}^+ \rightarrow \mathsf{DAC}$).
Thus, $\underline{\mathsf{B}}$ is the one and only candiate key. (Everything else would contain B and be a superkey.
$\mathsf{D} \rightarrow \mathsf{A}$: D is not a key or superkey and A is not prime. This breaks 3NF.

1NF: Domain of each attribute is an *elementary* type; that is, not a *set* or a *record structure.*

2NF: Whenever $\mathcal{X} \mapsto A$ is a functional dependency that holds in relation R and $A \notin \mathcal{X}$, then either

- A is *prime*, or
- $\mathcal{X}$ is not a proper subset of any key for R.

3NF: Whenever $\mathcal{X} \mapsto A$ is a functional dependency that holds in relation R and $A \notin \mathcal{X}$, then either

- A is *prime*, or
- $\mathcal{X}$ is a key or a super-key for R.

BCNF: Whenever $\mathcal{X} \mapsto A$ is a functional dependency that holds in relation R and $A \notin \mathcal{X}$, then

- $\mathcal{X}$ is a key or a super-key for R.

An attribute A is called *prime* if A is in any of the candidate keys.

Extra space.

Relax. Turn in your exam. Go home.